LEVEL III

(10)

DOT/FAA/RD-81/45, 2
MTR-81W120, 2

Automatic Traffic Advisory and Resolution Service (ATARS) Algorithms Including Resolution-Advisory-Register Logic

R.H. Lentz, W.D. Love, T.L. Signore
R.A. Tornese, A.D. Zeitlin

The MITRE Corporation
Metrek Division
McLean, Virginia 22102

June 1981

Volume 2-Sections 12 through 19; Appendixes

This document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161

US Department of Transportation
Federal Aviation Administration
Systems Research & Development Service
Washington, D.C. 20590

Technical Report Documentation Page

1. Report No. FAA-RD-81-45-2	2. Government Accession No. AD A104 148	3. Recipient's Catalog No.
4. Title and Subtitle Automatic Traffic Advisory and Resolution Service (ATARS) Algorithms Including Resolution-Advisory-Register Logic		5. Report Date June 1981
		6. Performing Organization Code
7. Author(s) R.H. Lentz, W.D. Love, T.L. Signore, R.A. Tornese, A.D. Zeitlin		8. Performing Organization Report No. MTR-81W120-2
9. Performing Organization Name and Address The MITRE Corporation Metrek Division 1820 Dolley Madison Boulevard McLean, Virginia 22102		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DOT-FA80WA-4370
12. Sponsoring Agency Name and Address U. S. Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington, D.C. 20590		13. Type of Report and Period Covered
		14. Sponsoring Agency Code ARD-240
15. Supplementary Notes		

16. Abstract

This document presents detailed computer algorithms for programming the Automatic Traffic Advisory and Resolution Service (ATARS). A major feature of this version of the ATARS algorithms is the capability to exchange resolution advisory information via the airborne Resolution Advisory Register (RAR). This provides for coordination of resolution advisories between ATARS and airborne collision avoidance systems and between adjacent ATARS sites in the absence of ground communication lines. The ground based ATARS computers use the surveillance data from the Discrete Address Beacon System (DABS) to provide properly equipped aircraft with traffic advisories and collision resolution advisories. These advisories are discretely delivered to the aircraft via the DABS data link. The ATARS algorithms are presented in two volumes rather than one large document in order to provide the algorithms in a more manageable form.

17. Key Words Collision Avoidance, Aircraft Separation Assurance, Traffic Advisory Service.	18. Distribution Statement Document is available to the public through the National Technical Information Service, Springfield, Virginia 22161

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

LIST OF ILLUSTRATIONS
(Continued)

LIST OF ILLUSTRATIONS
(Continued)

LIST OF ILLUSTRATIONS (Concluded)

12. MASTER RESOLUTION TASK

The Master Resolution Task utilizes the aircraft pair output data of the ATARS Detect Task to manage encounters and determine resolution advisories. Its functions are:

1. Cause resolution advisories to be issued when two requests for resolution advisories on any three consecutive scans have been generated by the detection logic or issue resolution advisories immediately when the detection logic has determined that a maneuvering aircraft is a threat.

2. Select the appropriate positive or negative resolution advisories for the pair using the Resolution Advisories Evaluation Routine (RAER). If existing resolution advisories prevent the selection of resolution advisories for this pair, attempt resolution later in the scan.

3. Recalculate resolution advisories if the advisories selected on the previous scan are incompatible with advisories selected by another source (a remote ATARS site or BCAS). Incompatibility could occur if two or more sources are selecting advisories on the same scan, and the sources do not have the capability of communicating with each other.

4. Calculate double dimension resolution advisories if either maneuvering aircraft's turn status changes so as to be counterproductive to the horizontal resolution advisory selected. Recalculate resolution advisories if the relative vertical velocity of the pair changes so as to indicate that the selected vertical resolution advisories are ineffective. Check for these conditions for two scans after resolution advisories are initially chosen or modified.

5. Select a resolution advisory for a controlled aircraft in conflict with an uncontrolled aircraft when the detection logic determines that the resolution advisory to the uncontrolled aircraft is not providing sufficient separation. Select a resolution advisory for a controlled aircraft whenever there is a multi-aircraft conflict regardless of the detection logic's determination of the necessity for a resolution advisory to the controlled aircraft.

6. Monitor the change in the resolution dimension miss distance and transition resolution advisories between positive and negative as the projected separation of the encounter changes.

7. Monitor the response of aircraft to ATARS positive single dimension resolution advisories and, if necessary, issue additional resolution advisories in the event of apparent non-response, as evidenced by a diminishing miss distance in the resolution dimension.

12.1 Overview

Table 12-1 is a high-level description of the major functions performed by the Master Resolution Task. The basic strategy of the task is to select resolution advisories for each conflict pair based on the status of all aircraft in a conflict cluster (as given in the Conflict Table). These advisories are recorded in the Pair Record and the effective resolution advisory is placed in each aircraft's Conflict Table Entry.

The resolution advisories in the Pair Record may be thought of as representing the desired resolution advisories for this pair. All of the desired resolution advisories for an aircraft are examined and the most severe resolution advisory in each dimension becomes the effective resolution advisory in that dimension in the Conflict Table Entry. As pairs go in and out of conflict, the effective resolution advisory for an aircraft may change severity (positive/negative) and/or dimension (vertical/horizontal).

Because the resolution logic interacts extensively with the Conflict Table and Pair Records, these data structures are discussed next. They are defined in pseudocode Section 3.5.

12.2 Conflict Table and Pair Record Data Structures

There are two types of data required by the Master Resolution Task for management of ATARS resolutions:

1. Inherently pairwise information, such as time at which resolution advisories were initiated or miss distance on previous scan

2. Multi-aircraft information concerning the entire cluster of conflicting aircraft

TABLE 12-1

MAJOR FUNCTIONS PERFORMED BY THE
MASTER RESOLUTION TASK

<u>Master Resolution Task</u>

1. Own ATARS site resolution responsibility

   - Other ATARS sites' resolution advisories adequacy test

2. Initial resolution advisory selection

   - Resolution Advisories Evaluation Routine

     - If unable to select advisories, delay resolution

3. Recalculate advisories if previous advisories are incompatible with advisories from another source

   - Resolution Advisories Evaluation Routine

     - If unable to select advisories, delay resolution

4. Recalculate advisories if the conflict geometry has changed detrimentally

   - Resolution Advisories Evaluation Routine

     - If unable to select advisories, continue present advisories

5. Positive/negative resolution advisory transition

   - Resolution Advisories Evaluation Routine

     - If unable to select advisories, continue present advisories

   - Vertical speed limit advisory evaluation

TABLE 12-1
(Concluded)

6. Controlled aircraft resolution advisory addition in controlled/uncontrolled pair

    - Resolution Advisories Evaluation Routine

        - If unable to select advisories, continue present advisories

7. Recalculate advisories if aircraft non-response is detected

    - Resolution Advisories Evaluation Routine

        - If unable to select advisories, continue present advisories

8. Resolution advisory posting to Pair Record and Conflict Table Entries

Each aircraft involved in a conflict has a pointer, CTPTR, in its system State Vector pointing to the head of a Conflict Table. The Conflict Table Head consists of the count of the number of aircraft in the cluster, pointers to the head of the next and previous Conflict Tables in the linked list of Conflict Tables, a flag that indicates whether any aircraft in this Conflict Table is in an ATARS seam, a pointer to the list of the Pair Records, and a pointer to the first of the Conflict Table Entries.

The Conflict Table Entries, one for each aircraft in a conflict, make up the body of the Conflict Table. The Conflict Table Entries are linked together to permit easy insertion or deletion of aircraft (although the table could be conceptually regarded as a simple array of Conflict Table Entries). The fields in each Conflict Table Entry are used to record information about the aircraft in relation to the conflict cluster, to record the effective vertical and horizontal resolution advisories (VMAN and HMAN) for each aircraft, and to record the advisories being displayed (VMAND and HMAND) after the most recent scan.

For every aircraft pair declared in conflict, a Pair Record is created and linked into the list of Pair Records for this Conflict Table. The Pair Record contains information on the particular encounter underway, the selected resolution advisories for the pair, pointers to the Conflict Table Entries of the aircraft involved and the identification of the ATARS function controlling the resolution of that pairwise conflict or an indication of BCAS control.

A Pair Record is also created when an aircraft receives a resolution advisory from BCAS or from a non-connected ATARS site. In this case, the identification of the other aircraft is set to a dummy value.

The interplay of the Conflict Table and Pair Records (as discussed in the following sections) permits:

1. The selection of resolution advisories based on the status of the entire conflict cluster under the multi-aircraft rules

2. The management of modifications to resolution advisories due to resolution advisory transitions in severity and dimension

The Pair Records and Conflict Table for a sample three-aircraft conflict are shown in Figure 12-1. Only a portion of each aircraft's State Vector is shown.

Both conflict pairs in the example have one aircraft, AC3, in common. Therefore, all three aircraft are placed in the same Conflict Table, CT1, which is pointed to by each aircraft's State Vector Conflict Table pointer, CTPTR. Each State Vector also points to its respective Conflict Table Entry by the Conflict Table Entry pointer, CTE. Since an aircraft may be in more than one Pair Record at a time, there is no pointer from the State Vector directly to the Pair Records. However, the list of Pair Records associated with this Conflict Table is pointed to from the Conflict Table Head using the pointer PLIST. A particular Pair Record may be pointed to from the Conflict Table Entry. A Pair Record that has a horizontal resolution advisory for an aircraft is pointed to by the ACIDH field in the Conflict Table Entry. Only one Pair Record is pointed to by ACIDH (ACIDV). The field MULTH in the Conflict Table Entry is a count of the number of Pair Records containing a horizontal resolution advisory for an aircraft. A value of MULTH (MULTV) greater than one indicates that more than one conflict pair is contributing to the effective horizontal (vertical) resolution advisory for an aircraft.

In the example shown in Figure 12-1, the first Pair Record records information for the conflict between AC1 and AC3. The selected advisories, turn left (L) and turn right (R), are recorded in the Pair Record and in the Conflict Table Entries. The Conflict Table Entries for AC1 and AC3 point to the first Pair Record, PR1, as containing the horizontal resolution advisories for these two aircraft.

The second conflict involves AC2 and AC3. The selected resolution advisories climb (C) and descend (D) for this pair are recorded in the second Pair Record, PR2, and in the Conflict Table Entries for AC2 and AC3.

The field NCON in the Conflict Table Entries is a count of the number of conflicts in which an aircraft is involved.

12.3 Selection of Resolution Advisories for a Conflict Pair

The Master Resolution Task uses the output of the Detect Task to determine if resolution advisories must be calculated for a pair of aircraft. Master resolution may determine that resolution advisories are not required, that this is the first time

AIRCRAFT STATE VECTORS

(AC 1)	
CTPTR	CT1
CTE	CTE1
(AC 2)	
CTPTR	CT1
CTE	CTE2
(AC 3)	
CTPTR	CT1
CTE	CTE3

CONFLICT TABLF HEAD

(CT1)	
FCTE	CTE1
NEXTCT	NULL
PREVCT	NULL
NAC	3
PLIST	PR1
SEAM	0

PAIR RECORDS

	(PR1)	(PR2)
NXTPR	PR2	NULL
ATSID	OWNID	OWNID
HDOFF		
SECTID		
PIFR	0	0
PMD	MD2	MD2
POSCMD	$POS	$POS
PVMD	ALT	ALT
PWISF	1	1
TSTART	CTIME	CTIME
CMDFL1		
EHMAN1		
EVMAN1		
INTR1		
MVT1		
PAC1	CTE1	CTE2
PHMAN1	L	
PVMAN1		C
SEND1	1	1
TRKID1		
CMDFL2		
EHMAN2		
EVMAN2		
INTR2		
MVT2		
PAC2	CTE3	CTE3
PHMAN2	R	
PVMAN2		D
SEND2	1	1
TRKID2		
MVDONE		
MVRAIT		
MVVRZ		

CONFLICT TABLE ENTRIES

	NXTCTE	ACID	ACIDH	ACIDV	HMAN	HMAND	MULTH	MULTV	NCON	REMFLG	VMAN	VMAND
(CTE1)	CTE2	AC1	PR1		L	L	1		1	0		
(CTE2)	CTE3	AC2		PR2				1	1	0	C	C
(CTE3)	NULL	AC3	PR1	PR2	R	R	1	1	2	0	D	D

FIGURE 12-1
DATA STRUCTURES FOR A SAMPLE THREE-AIRCRAFT CONFLICT

resolution advisories are required or that resolution advisories should be recomputed for the pair. Each of these possibilities is discussed in the following sections.

Before the Master Resolution Task can determine if resolution advisories are required this scan, it must determine if this ATARS site is responsible for the pair. If the RAREQ flag is set in the Encounter List entry, then own site is responsible. However, if the Seam Pair Task has flagged this site as provisionally responsible, then master resolution examines the resolution advisories being given to the DABS aircraft in the pair from other, higher-priority ATARS sites. If those advisories are determined to be adequate to resolve the conflict, then own site is not responsible for the pair. If they are determined not adequate, then own site takes responsibility for the pair.

12.3.1 Initiation of Resolution Advisories

Because ATARS decisions are based on tracked information which is subject to random fluctuations from scan to scan, it is desirable to incorporate logic to reduce false alarms when dealing with resolution advisories. Incorporating a rule which requires the conditions for issuing resolution advisories to be satisfied on two consecutive scans normally could prevent unnecessary resolution advisories because of errors on a single scan. But this rule can also lead to late alarms. If on one scan the calculations require resolution advisories, but on the second scan they do not (because of random errors) when they should, then it would require two additional scans to fully declare the conflict, and a late resolution would occur. To alleviate this problem, a rule is implemented which will issue resolution advisories if the Detect Task requires resolution advisories on any two of three consecutive scans.

This two-out-of-three rule is implemented through the use of a conflict control variable, POSCMD. When a request for resolution advisories is generated for a given pair, a Pair Record is created and POSCMD is initialized. POSCMD is then updated according to the transition logic in Table 12-2. POSCMD is updated on each scan that the Master Resolution Task is called for a conflict pair, until POSCMD reaches a value indicating that resolution advisories should be computed. The maneuvering target threat flag, MTTFLG, indicates an immediate need for resolution advisories. If MTTFLG is set (Section 8), then the normal transition sequence is bypassed and resolution advisories are calculated immediately.

TABLE 12-2

POSCMD TRANSITION LOGIC

PREVIOUS POSCMD[1]	NEW VALUE OF POSCMD BASED ON VALUE OF FLAGS FOR CURRENT SCAN		
	MTTFLG Flag		
	Set	Not set	
		CMDFLG Flag	
		Set	Not Set
$NOTSET[2]	$RANEC	$ONEHIT	$NORA
$ONEHIT	$RANEC	$RANEC	$ONEMIS
$ONEMIS	$RANEC	$RANEC	$NORA

[1]POSCMD takes on additional values after resolution advisories are selected. See Appendix B.

[2]$NORA -	Conflict detected on only one out of three scans. Resolution advisories are not needed for this pair. The pair may be deleted.
$NOTSET -	Initial value, POSCMD not set.
$ONEHIT -	Conflict detected on one scan. Resolution advisories not yet necessary.
$ONEMIS -	Conflict detected on one scan, no conflict detected on the next scan. Resolution advisories not yet necessary.
$RANEC -	A conflict has been detected on two out of three scans or the immediate need for resolution advisories has been detected. Resolution advisories should be computed for this pair.

The Detect Task determines the need for resolution advisories and sets the CMDFLG and MTTFLG accordingly. The Seam Pair Task evaluates the site's resolution responsibility for the pair. When no resolution is performed, Resolution Deletion Task or Conflict Pair Cleanup Task handles the updating of POSCMD. If POSCMD reaches a state indicating no resolution advisory is necessary, the pair is declared to be not in conflict and the Pair Record is deleted (Section 15). The Conflict Table Entries may be deleted if the aircraft are in no other conflicts.

12.3.1.1 Initial Resolution Advisory Selection

When master resolution first determines that resolution advisories should be selected for a pair of aircraft in conflict, it calls the Resolution Advisories Evaluation Routine (RAER) to select the actual advisories. Three parameters are passed to the Resolution Advisories Evaluation Routine from master resolution along with the identification of the subject conflict pair. One parameter indicates whether single or double dimension resolution advisories are desired.

The second parameter indicates that the Master Resolution Task is calling the Resolution Advisories Evaluation Routine. This indicates to RAER that the complete resolution logic should be performed. This is in contrast to when RAER is called by the Conflict Resolution Data Task. These differences are explained in Section 13.

The third parameter is passed indirectly to the Resolution Advisories Evaluation Routine. This is an indication of whether the controlled aircraft is to be maneuvered. A resolution advisory is always selected for an ATARS-equipped uncontrolled aircraft in a conflict pair when it is not in a final approach zone. If either aircraft is an ATARS-equipped controlled aircraft, a resolution advisory is selected for that aircraft only if the PIFR flag in the Pair Record has been set by master resolution.

Master resolution sets the PIFR flag in the Pair Record if the detection logic has indicated that any controlled aircraft in this pair should receive resolution advisories. The detection logic indicates this by setting the IFRFLG.

Master resolution will also set the PIFR flag if the current conflict pair contains an ATARS-equipped controlled aircraft and is part of a multi-aircraft conflict. If an ATARS-equipped aircraft is in a multi-aircraft cluster, it is desirable to actively maneuver the aircraft in resolving the conflict, rather than counting on two or more other aircraft to resolve the conflict.

12.3.2 Resolution Advisory Change Logic

Subsequent to the first scan that resolution advisories are selected, the severity (positive/negative), resolution dimension (horizontal/vertical) and even the number (single/double dimension) of resolution advisories for a conflict pair may change from scan to scan. The various conditions under which the resolution advisories may change are described in the following sections.

12.3.2.1 Recomputation Because of Incompatible Resolution Advisories

Resolution advisories may be selected for an aircraft pair at one ATARS site at the same time that resolution advisories are being selected for one or both of the aircraft by BCAS or by another non-connected ATARS site. If this situation occurs it is possible for the resolution advisories selected by the two different sources to be incompatible. If the resolution advisories from BCAS or the other ATARS site are accepted by the aircraft before the resolution advisories from the local site are uplinked, then the advisories from the local site will be rejected by the ATARS avionics. This condition will be recognized by the RAR Processing Task, which will delete the advisories in the Pair Record and set the conflict control variable, POSCMD, to indicate that the resolution advisories from this site must be recalculated. The setting of POSCMD will also indicate if single or double dimension resolution advisories were selected by this site.

The first check done by the Master Resolution Task when resolution advisories were given previously is to determine if they must be recomputed because of incompatibility with resolution advisories from other ATARS sites or BCAS. If RAER is called to recompute advisories and it is unable to select new resolution advisories, resolution is delayed.

12.3.2.2 Validation of Resolution Advisories

The horizontal turn status of each maneuvered aircraft and the relative vertical velocity of the aircraft pair are factors in selecting resolution advisories for a conflict pair. To ensure that the turn status of the maneuvered aircraft or the relative vertical velocity of the pair has not changed in a way detrimental to the selected resolution advisories, a validation of the PSEP modeling assumptions is performed within two scans of selecting single dimension resolution advisories. This logic is referred to as the PSEP model validation logic.

For two scans after horizontal-only resolution advisories are selected, master resolution checks for the turn status of either aircraft changing from its status on the scan in which resolution advisories were selected. If the turn status has changed and the current turn status is detrimental to the selected resolution advisories, then resolution advisories are recalculated. RAER is called and double dimension resolution advisories are requested. Table 12-3 shows the conditions under which resolution advisories are recalculated because of turn status changes.

If the relative vertical velocity changes significantly within two scans of selecting vertical-only resolution advisories, and the chosen advisories are ineffective based on the current relative vertical velocity, then resolution advisories are recalculated. RAER is called and double dimension resolution advisories are requested. Table 12-4 shows the conditions under which changes in the relative vertical velocity of the pair cause a recalculation of resolution advisories. If RAER is unable to select resolution advisories, the previously selected advisories continue to be given.

12.3.2.3 Controlled/Uncontrolled Conflict Pair

When an ATARS-equipped controlled aircraft and an equipped uncontrolled aircraft are declared in conflict by the detection logic, normally the uncontrolled aircraft is maneuvered without maneuvering the controlled aircraft. This is accomplished by using larger detection thresholds to determine the need for the uncontrolled aircraft's advisory than are used to determine the need for the controlled aircraft's advisory. However, if it is determined on a later scan that the controlled aircraft should be maneuvered, RAER is called to compute advisories for both aircraft. The PIFR flag is set in the Pair Record to indicate that the controlled aircraft should receive an advisory.

If the Resolution Advisories Evaluation Routine is able to compute advisories for both aircraft, then the PIFR flag remains set in the Pair Record. If RAER is not able to compute an advisory for the controlled aircraft, PIFR is reset and resolution is delayed. If this pair is being processed by

TABLE 12-3

HORIZONTAL TURN STATUS CHANGES SINCE RESOLUTION ADVISORY SELECTION THAT MAY CAUSE RECOMPUTATION

HORIZONTAL RESOLUTION ADVISORY	CURRENT TURN STATUS	TURN STATUS WHEN RESOLUTION ADVISORY SELECTED		
		$STRNGLFT	$STRNGRGT	ALL OTHERS
$TL[1]	$STRNGLFT	$FALSE[2]	$FALSE	$FALSE
OR	$STRNGRGT	$TRUE	$FALSE	$TRUE
$DTR	ALL OTHERS	$TRUE	$FALSE	$FALSE
$TR	$STRNGLFT	$FALSE	$TRUE	$TRUE
OR	$STRNGRGT	$FALSE	$FALSE	$FALSE
$DTL	ALL OTHERS	$FALSE	$TRUE	$FALSE
$NORES	ALL VALUES	$FALSE	$FALSE	$FALSE

[1]Complete description provided in Appendix B.
[2]$FALSE - advisories do not need to be recalculated.
$TRUE - advisories do need to be recalculated.

TABLE 12-4

RELATIVE VERTICAL VELOCITY CHANGES SINCE
RESOLUTION ADVISORY SELECTION THAT MAY
CAUSE RECOMPUTATION

SELECTED RESOLUTION ADVISORIES		CURRENT RELATIVE VERTICAL VELOCITY MINUS RELATIVE VELOCITY AT RESOLUTION ADVISORY SELECTION TIME		
ACID1	ACID2	LESS THAN -ZDRTH[1]	GREATER THAN OR EQUAL TO -ZDRTH AND LESS THAN OR EQUAL TO ZDRTH	GREATER THAN ZDRTH
\$CL[2], \$DDES, \$NORES LIMIT DESCEND VSL's	\$DES, \$DCL, \$NORES LIMIT CLIMB VSL's	\$FALSE[3]	\$FALSE	\$TRUE
\$DES, \$DCL, \$NORES \$FALSE LIMIT CLIMB VSL's	\$CL, \$DDES, \$NORES LIMIT DESCEND VSL's	\$TRUE	\$FALSE	

[1]ZDRTH = MAX(MVZDM, MVZDF*PREC.MVVRZ)
MVZDF = 0.2
MVZDM = 300 fpm

[2]Complete description provided in Appendix B.

[3]\$FALSE - advisories do not need to be recalculated
TRUE - advisories do need to be recalculated

Normal Master Resolution, then resolution is delayed by setting an Encounter List flag appropriately so that this pair will be processed by Delayed Master Resolution. This will give the Resolution Advisories Evaluation Routine a second chance on this same scan to try to compute an advisory for the controlled aircraft. The advisory to the uncontrolled aircraft should not be deleted from the Pair Record even if an advisory to the controlled aircraft can not be added.

12.3.2.4 Positive/Negative Resolution Advisory Transition

Resolution advisories are monitored to determine if a negative-to-positive transition is required or a positive-to-negative transition is allowed. If either transition may occur, new resolution advisories are selected and entered in the Pair Record. When positive resolution advisories are selected they must continue for at least TSCMD seconds before a transition may occur. If positive resolution advisories have been issued in both planes for the given pair and the resolution advisories in one plane transition to negatives, the resolution advisories in the other plane are deleted.

Horizontal resolution advisories are checked for possible transition by comparing the miss distance calculated by the Detect Task against the negative horizontal resolution advisory threshold. If the miss distance is less than the threshold, then positive resolution advisories are needed. Otherwise, negative resolution advisories are acceptable. The normal negative horizontal resolution advisory threshold is modified (increased) if either aircraft is turning.

Vertical resolution advisories are checked for possible transition by comparing the current vertical separation against the negative vertical resolution advisory threshold. If the current altitude separation is greater than the threshold, and the aircraft are diverging vertically, then positive vertical resolution advisories may transition to negatives.

If the current altitude separation is less than the threshold, then an additional check is performed before requiring negative verticals to transition to positives. The current altitude separation must be less than a parameter (ATBZP) percent of the threshold before the transition to positives is required.

If the Pair Record contains resolution advisories of a different severity (positive/negative) than those determined necessary by the transition logic, then the Resolution Advisories Evaluation

Routine is called to select new resolution advisories. The one exception to this rule is if negative vertical resolution advisories are deemed acceptable and negative vertical resolution advisories are already in the Pair Record. Then, the vertical speed limit (VSL) evaluation logic is performed.

12.3.2.5 Non-responding Aircraft Logic

Additional logic provides for selecting double dimension resolution advisories when either or both aircraft have not adequately responded to the positive single dimension resolution advisories previously computed. A test for non-response to resolution advisories is performed until the aircraft have had a chance to respond to the resolution advisories. Response is not evaluated until TRECOM seconds after resolution advisory selection. Non-response to resolution advisories is inferred if the miss distance in the resolution dimension decreases from one scan to the next. RAER is called and double dimension resolution advisories are requested. If RAER is unable to select new advisories, the previously selected advisories are not deleted.

12.3.3 Resolution Advisories in the Pair Record and Conflict Table

After calling RAER to select resolution advisories, master resolution must record the advisories selected, or handle the pair properly if no advisories were selected.

When RAER returns resolution advisories to the Master Resolution Task, they are first compared with the advisories that are currently in the Pair Record (if any exist). If this is the initial selection of advisories, or any of the advisories have changed from the previous scan, then the new advisories are stored in the Pair Record. Also, the POSCMD field is set appropriately, TSTART is set to the current time, the current horizontal and vertical miss distances are saved and the advisories are flagged to be sent to the aircraft. To facilitate the PSEP model validation logic, the turn status of both aircraft and relative vertical velocity are saved. If the new advisories were selected because of the PSEP model validation logic, then MVDONE is set to $TRUE. Otherwise, it is set to $FALSE.

After storing the advisories in the Pair Record, both aircraft's Conflict Table Entries are updated. The fields MULTH and MULTV are set to the number of conflict pairs contributing to the horizontal and vertical advisories for each aircraft. The Conflict Table Entry fields HMAN and VMAN are set to the effective horizontal and vertical maneuvers. The effective maneuvers are determined by examining every Conflict Table Entry with an advisory for either aircraft and combining the advisories using the logic in Tables 12-5 and 12-6. It should be noted that the effective vertical resolution advisory determination logic shown in Table 12-6 is used only for the Conflict Table Entry field VMAN. The field VMAND may take on additional values. These additional values are determined by the RAR Processing Task (Section 5.2).

Any advisories currently in the Pair Record remain in the Pair Record when RAER is unable to select new resolution advisories. This is true if the pair is being processed by either Master Resolution Normal or Delayed.

12.4 Pseudocode for Master Resolution Task

The high- and low-level pseudocode for the Master Resolution Task is presented in this section.

The low-level pseudocode uses a shorthand pointer notation. Rather than use a pointer name pointing to a data structure, the effective pointer is used in place of the name of the data structure. For example, on page 12-P11, the notation TPREC.acl.PAC.ACID is used. This is shorthand for TPREC(pointer) pointing to PREC.acl.PAC(pointer) pointing to CTENTRY.ACID (note Appendix C).

Another convention is used within LOOPs. The current value of the variable that is the index of the LOOP is denoted in one of two ways. If the variable has a number suffix (1 or 2), the suffix is dropped within the LOOP. If the variable does not normally have a suffix, it is given the prefix T (e.g., TPREC is used for PREC).

TABLE 12-5

EFFECTIVE HORIZONTAL RESOLUTION ADVISORY DETERMINATION LOGIC

		HORIZONTAL RESOLUTION ADVISORY BEING ADDED						
		$TL	$TR	$DTR	$DTL	$DTLDTR	$NULLRES	$NORES
CURRENT	$TL[1]	$TL	*	$TL	*	*	$TL	$TL
EFFECTIVE	$TR	*	$TR	*	$TR	*	$TR	$TR
HORIZONTAL	$DTR	$TL	*	$DTR	$DTLDTR	$DTLDTR	$DTR	$DTR
RESOLUTION	$DTL	*	$TR	$DTLDTR	$DTL	$DTLDTR	$DTL	$DTL
ADVISORY	$DTLDTR	*	*	$DTLDTR	$DTLDTR	$DTLDTR	$DTLDTR	$DTLDTR
	$NULLRES	$TL	$TR	$DTR	$DTL	$DTLDTR	$NULLRES	$NORES
	$NORES	$TL	$TR	$DTR	$DTL	$DTLDTR	$NORES	$NORES

[1]Complete description provided in Appendix B. The symbol * indicates an incompatible combination.

TABLE 12-6

EFFECTIVE VERTICAL RESOLUTION ADVISORY DETERMINATION LOGIC

		VERTICAL RESOLUTION ADVISORY BEING ADDED												
		$CL	$DES	$DDES	$DCL	$DCLDDES	$LDES2K	$LCL2K	$LDES1K	$LCL1K	$LDES500	$LCL500	$NULLRES	$NORES
CURRENT	$CL[1]	$CL	*	$CL	*	*	$CL	*	$CL	*	$CL	*	$CL	$CL
EFFECTIVE	$DES	*	$DES	*	$DES	*	*	$DES	*	$DES	*	$DES	$DES	$DES
VERTICAL	$DDES	$CL	*	$DDES	$DCLDDES	$DCLDDES	$DDES	$DCLDDES	$DDES	$DCLDDES	$DDES	$DCLDDES	$DDES	$DDES
RESOLUTION	$DCL	*	$DES	$DCLDDES	$DCL	$DCLDDES	$DCLDDES	$DCL	$DCLDDES	$DCL	$DCLDDES	$DCL	$DCL	$DCL
ADVISORY	$DCLDDES	*	*	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES	$DCLDDES
	$LDES2K	$CL	*	$DDES	$DCLDDES	$DCLDDES	$LDES2K	$DCLDDES	$LDES1K	$DCLDDES	$LDES500	$DCLDDES	$LDES2K	$LDES2K
	$LCL2K	*	$DES	$DCLDDES	$DCL	$DCLDDES	$DCLDDES	$LCL2K	$DCLDDES	$LCL1K	$DCLDDES	$LCL500	$LCL2K	$LCL2K
	$LDES1K	$CL	*	$DDES	$DCLDDES	$DCLDDES	$LDES1K	$DCLDDES	$LDES1K	$DCLDDES	$LDES500	$DCLDDES	$LDES1K	$LDES1K
	$LCL1K	*	$DES	$DCLDDES	$DCL	$DCLDDES	$DCLDDES	$LCL1K	$DCLDDES	$LCL1K	$DCLDDES	$LCL500	$LCL1K	$LCL1K
	$LDES500	$CL	*	$DDES	$DCLDDES	$DCLDDES	$LDES500	$DCLDDES	$LDES500	$DCLDDES	$LDES500	$DCLDDES	$LDES500	$LDES500
	$LCL500	*	$DES	$DCLDDES	$DCL	$DCLDDES	$DCLDDES	$LCL500	$DCLDDES	$LCL500	$DCLDDES	$LCL500	$LCL500	$LCL500
	$NULLRES	$CL	$DES	$DDES	$DCL	$DCLDDES	$LDES2K	$LCL2K	$LDES1K	$LCL1K	$LDES500	$LCL500	$NULLRES	$NORES
	$NORES	$CL	$DES	$DDES	$DCL	$DCLDDES	$LDES2K	$LCL2K	$LDES1K	$LCL1K	$LDES500	$LCL500	$NORES	$NORES

[1]Complete description provided in Appendix B. The symbol * indicates an incompatible combination.

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

```
---------------------------------------------------------------------------------

STRUCTURE MRPARM

  GROUP res_adv_computation
    FLT ALPC          <lower limit of positive controlled airspace used to
                        select positive resolution advisory altitude threshold>
    FLT ALUH          <lower limit of ultra high altitude airspace used to
                        select positive resolution advisory altitude threshold>
    FLT ASEPH         <high altitude positive resolution advisory threshold>
    FLT ASEPIL        <low altitude positive resolution advisory threshold for
                        controlled/uncontrolled and controlled/controlled>
    FLT ASEPL         <low altitude pos res adv threshold for uncontrolled pair>
    FLT ASEPU         <ultra high altitude positive res adv threshood>

  GROUP res_adv_recomputation
    FLT ATBZP         <Advisory Transition Buffer Zone Percentage>
    FLT MVZDF         <PSEP model validation ZD factor>
    FLT MVZDM         <PSEP model validation ZD maximum>
    FLT TRECOM        <delay before positive res adv may be checked for response>
    FLT TSCND         <delay before positive res adv may be checked for
                        transition to negatives>

  GROUP miscellaneous
    INT APAIR         <number of AC in a conflict pair>
    FLT TVALID        <number of scans when PSEP model validation
                        logic performed>
```

```
-----------------------------------------------------------------------------------

 GROUP logic_tables
   BIT DETRIMH(5,7,7)       <PSEP model validation logic table for detrimental
                              turn state changes vs. previous turn states
                              and previous selected res adv:
                              (selected hor res adv, current turn state, turn state
                              when resolution advisories selected)>

   BIT DETRIMV(11,11,3)     <PSEP model validation logic table for detrimental
                              relative vertical velocity changes:
                              (vert res adv for first AC, vert res adv for sec AC,
                              difference between current relative vertical velocity
                              and relative vertical velocity when res adv selected)>

   INT EFFHRA(7,7)          <effective horizontal resolution advisory selection:
                              (res adv to be added, current effective res adv)>

   INT EFFVRA(13,13)        <effective vertical resolution advisory selection:
                             (res adv to be added, current effective res adv)>

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

STRUCTURE MRVBL

  GROUP logic_path
     BIT MRNCAP             <Master Resolution called RAER when true, Conflict
                               Resolution Data Task called RAER when false>
     BIT RASELECT           <resolution advisories selected this scan>
     BIT RECALC             <RA's are to be or have been recalculated this scan>
     BIT SNGDIM             <single dimension RA's preferred when true, double dim when
                               false>

  GROUP other_site
     INT OSHMAN             <horizontal res adv from other ATARS non-connected sites>
     INT OSVMAN             <vertical res adv from other ATARS non-connected sites>
     FLT PSEPSQ             <squared 3-D separation>

  GROUP pointer
     PTR ELENTRY            <encounter list entry>
     PTR PREC               <subject pair record>
     PTR RADSPTR            <selected resolution advisory set>
     PTR TACID              <temporary state vector pointer>
     PTR TPREC              <temporary pair record pointer>

  GROUP res_adv_thr
     FLT ASEP               <altitude separation threshold>
     FLT NDTHR              <negative horizontal res adv threshold>

ENDSTRUCTURE;
```

```
---------------------------------------------------------------------------------

STRUCTURE TRADS              <Resolution Advisory Data Structure>

  GROUP pointers

     PTR NXTADV              <next RADS in list>

  GROUP advisory_components

     INT H1                  <horizontal component of AC 1's res adv>

     INT H2                  <horizontal component of AC 2's res adv>

     INT V1                  <vertical component of AC 1's res adv>

     INT V2                  <vertical component of AC 2's res adv>

  GROUP read-only_flags

     BIT CMDED_CMDED         <advisory set maneuvers both AC>

     BIT CMDED_UNCMDED       <advisory set maneuvers first AC in pair>

     BIT HORIZ               <advisory in horiz dimension when true>

     BIT SINGLE              <advisory is one dimension only when true>

     BIT UNCMDED_CMDED       <advisory maneuvers second AC in pair>

     BIT VERT                <advisoy in vertical dimension when true>

  GROUP read/write_flags

     BIT BELOW1000           <descend res adv must be changed to negative>

     BIT NEGATIVE            <negative res adv provide sufficient separation>

  GROUP sep_matrix_indices

     INT INDEX1              <PSEP (HMD) index for first AC's horizontal advisory>

     INT INDEX2              <PSEP (HMD) index for second AC's horizontal advisory>

     INT INDEX3              <PSEP (VMD) index for vertical level>

     PTR MATPTR              <pointer to separation matrices to be used with
                                 this resolution advisory set>



----------------------- MASTER RESOLUTION LOCAL VARIABLES ------------------------
```

```
-----------------------------------------------------------------------------------

  GROUP other-info
     INT DOMVALUE              <computed value of this advisory's features down
                                   to domino features>
     BIT FEATBITS(25)          <one bit for each of 25 features>
     INT VALUE                 <computed relative value of this advisory>

ENDSTRUCTURE;
```

```
----------------------- MASTER RESOLUTION LOCAL VARIABLES -------------------------
```

```
------------------------------------------------------------------------------------
TASK MASTER_RESOLUTION
   IN (encounter list entry)
   OUT (pair record and conflict table entries with resolution advisories);

     LOOP:
          Get next pair requiring resolution from this sectors encounter list;
     EXITIF (no pairs remain);
          PERFORM conflict_pair_record_determination;
          IF (own ATARS site is provisionally responsible for this pair)
               THEN PERFORM other_ATARS_site_resolution_advisory_adequacy_test;
               ELSE:
          IF (resolution advisories are required)
               THEN PERFORM resolution_advisory_necessity_check_and_variable_
                                                              initialization;
                    IF (need for resolution advisories exists on this scan)
                         THEN CALL ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION;
                              SET flag to indicate that Resolution Advisories
                                   Evaluation Routine called from Master Resolution;
                              CLEAR flag indicating resolution advisories have
                                   been selected this scan;
                              IF (computing resolution advisories for the first time)
                                   THEN PERFORM initial_resolution_advisory_
                                                                   selection;
                                   ELSE PERFORM previous_resolution_advisory_
                                                              modification_tests;
                              IF (resolution advisories have been selected in pair
                                        record this scan)
                                   THEN PERFORM resolution_advisory_posting_from_
                                                      pair_record_to_conflict_table;
                                   ELSE:     <resolution is delayed or unchanged>
                         ELSE:     <resolution advisories not yet needed>
               ELSE:    <no resolution performed>
     ENDLOOP;

END MASTER_RESOLUTION;
-----------------------    MASTER RESOLUTION HIGH-LEVEL LOGIC    -----------------------
```

```
------------------------------------------------------------------------------------

TASK MASTER_RESOLUTION
   IN (ELENTRY)
   OUT (PREC);

     LOOP:
          Get next aircraft pair with ELENTRY.RAREQ EQ $TRUE;
     EXITIF (no pairs remain);
          PERFORM conflict_pair_record_determination;
          IF (ELENTRY.RAPROV EQ $TRUE)
               THEN PERFORM other_ATARS_site_resolution_advisory_adequacy_test;
               ELSE:
          IF (ELENTRY.RAREQ EQ $TRUE)
               THEN PERFORM resolution_advisory_necessity_check_and_variable_
                                                                initialization;
                    IF ((PREC.POSCMD NE $ONEMIS) OR (PREC.POSCMD NE $ONEHIT))
                         THEN CALL ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION
                                   IN (ACID1, ACID2)
                                   OUT (ASEP);
                              HRNCAP = $TRUE;
                              RASELECT = $FALSE;
                              IF (PREC.POSCMD EQ $RANEC)
                                   THEN PERFORM initial_resolution_advisory_
                                                                      selection;
                                   ELSE PERFORM previous_resolution_advisory_
                                                                 modification_tests;
                              IF (RASELECT EQ $TRUE)
                                   THEN PERFORM resolution_advisory_posting_from_
                                                        pair_record_to_conflict_table;
                                   ELSE:    <resolution is delayed or unchanged>
                         ELSE:      <RAs not yet needed>
               ELSE:   <no resolution performed>
     ENDLOOP:

END MASTER_RESOLUTION;

---------------------- MASTER RESOLUTION LOW-LEVEL LOGIC ------------------------
```

```
--------------------------------------------------------------------------------
PROCESS conflict_pair_record_determination;

     <Determine which pair record is associated with the subject pair and set
      a pointer to that pair record.>

     CLEAR subject pair record pointer;
     Locate conflict table pointed to by aircraft in the encounter list entry;
     IF (only two aircraft in conflict table)
          THEN only pair record is the subject pair record;
          ELSE LOOP;
                    Get next pair record of conflict table;
               EXITIF (no more pair records OR subject pair record found);
                    IF (both AC from encounter list entry are in this pair)
                         THEN save this pair record as the subject pair record;
                         ELSE;
               ENDLOOP;

END conflict_pair_record_determination;
```

```
--------------------------------------------------------------------------------

PROCESS conflict_pair_record_determination;

     <PREC is used throughout this task in place of the local variable
       MRVBL.PREC.  This notation for the pointer PREC should not be
       confused with the pair record data structure, PREC.>

     PREC = $NULL;
     IF (ACID1.CTPTR.NAC EQ APAIR)
          THEN PREC = ACID1.CTPTR.PLIST;
          ELSE LOOP:
                    Get next pair record of conflict table;
               EXITIF (no more pair records OR (PREC NE $NULL));
                    IF (((ACID1 EQ TPREC.ac1.PAC.ACID) OR
                              (ACID1 EQ TPREC.ac2.PAC.ACID)) AND
                              ((ACID2 EQ TPREC.ac1.PAC.ACID) OR
                              (ACID2 EQ TPREC.ac2.PAC.ACID)))
                         THEN PREC = TPREC;
                         ELSE;
               ENDLOOP;

END conflict_pair_record_determination;
```

```
-------------------------------------------------------------------------------------
PROCESS other_ATARS_site_resolution_advisory_adequacy_test;

     <The subject pair is being handed-off to another site or being handed-off
      from another site.  If the ATARS equipped aircraft in the subject
      ATARS/ATCRBS pair has a resolution advisory from another higher priority
      site, determine if that resolution advisory is adequate to resolve the
      subject conflict pair.  If it is, then this site does not take
      responsibility for the pair.>

     PERFORM other_ATARS_sites_resolution_advisories_determination;

     IF (current scan's resolution advisories from other sites NE
                   resolution advisories from other sites from previous scan)
          THEN store resolution advisories from other sites in pair record;
               CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION;
               IF (adequate separation is modeled)
                    THEN SET horizontal and vertical resolution advisories in pair
                              record from own site to no resolution advisory;
                         SET flag in pair record to send resolution advisory
                              to aircraft;
                         SET pair record timer to current scan time;
                         CLEAR resolution advisory indication in encounter list;
                    ELSE;
          ELSE;     <no resolution advisories from other sites, or
                         they are the same as previously checked>

END other_ATARS_site_resolution_advisory_adequacy_test;
```

```
-----------------------------------------------------------------------------------

PROCESS other_ATARS_site_resolution_advisory_adequacy_test;

     PERFORM other_ATARS_sites_resolution_advisories_determination;

     IF ((OSHMAN NE PREC.EHMAN) OR (OSVMAN NE PREC.EVMAN))
          THEN PREC.EHMAN = OSHMAN;
               PREC.EVMAN = OSVMAN;
               CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION
                       IN (OSHMAN,OSVMAN,$NULLRES,$NULLRES,ACID1,ACID2)
                       OUT (PSEPSQ);
               IF (PSEPSQ GE RESADV.SEP1)
                    THEN PREC.ac1.PHMAN = $NORES;
                         PREC.ac2.PHMAN = $NORES;
                         PREC.ac1.PVMAN = $NORES;
                         PREC.ac2.PVMAN = $NORES;
                         PREC.ac1.SEND = $TRUE;
                         PREC.ac2.SEND = $TRUE;
                         PREC.TSTART = SYSVAR.CTIME;
                         ELENTRY.RAREQ = $FALSE;
                    ELSE;
          ELSE;

END other_ATARS_site_resolution_advisory_adequacy_test;
```

```
PROCESS resolution_advisory_necessity_check_and_variable_initialization;

    <The conflict control variable is updated each scan until the need for a
     resolution advisory is initially determined.  Then, the various states of the
     resolution advisories (negative, positive, double dimension) are monitored
     by the conflict control variable.>

    IF (conflict control variable shows that resolution advisories have not been
                given previously)
        THEN CALL CONFLICT_CONTROL_VARIABLE_UPDATE;
        ELSE:

END resolution_advisory_necessity_check_and_variable_initialization;
```

```
-----------------------------------------------------------------------------------

PROCESS resolution_advisory_necessity_check_and_variable_initialization;

    IF ((PREC.POSCMD EQ $NOTSET) OR (PREC.POSCMD EQ $ONEHIT) OR
            (PREC.POSCMD EQ $ONEMIS))
        THEN CALL CONFLICT_CONTROL_VARIABLE_UPDATE
                    IN (ELENTRY.CMDFLG, ELENTRY.HTTFLG)
                    INOUT (PREC.POSCMD);
        ELSE;

END resolution_advisory_necessity_check_and_variable_initialization;
```

```
------------------------------------------------------------------------------------

PROCESS initial_resolution_advisory_selection;

     <Select initial resolution advisories. Determine if a controlled
      aircraft should receive a resolution advisory and set PIFR in the
      pair record appropriately. If resolution advisories are selected,
      store them in the pair record. Otherwise, delay resolution
      for the pair.>

     IF ((controlled AC in conflict pair) AND ((detection logic indicates that
               controlled AC should receive a resolution advisory) OR (this conflict
               pair is part of a multi-aircraft conflict))
          THEN indicate in pair record that controlled AC should receive a resolution
                    advisory;
          ELSE;
     SET flag to indicate single dimension resolution advisories preferred;
     CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;   <select the res advisories>

     IF (resolution advisories selected for the pair)
          THEN PERFORM resolution_advisories_store_in_pair_record;
          ELSE SET conflict control variable to low value so that initial selection
                    of resolution advisories will be attempted either later this
                    scan or within the next two scans if the need for resolution
                    advisories is again detected by the Detection Task;
               CLEAR flag in the pair record indicating controlled aircraft should
                    receive a resolution advisory;
               IF (this pair is flagged for normal resolution)
                    THEN flag this pair for delayed resolution;
                    ELSE;

END initial_resolution_advisory_selection;
```

```
--------------------------------------------------------------------------------
PROCESS initial_resolution_advisory_selection;

    IF (((ACID1.CUNC EQ $TRUE) OR (ACID2.CUNC EQ $TRUE)) AND
            ((ELENTRY.IFRFLG EQ $TRUE) OR (ACID1.CTPTR.NAC GT APAIR)))
        THEN PREC.PIFR = $TRUE;
        ELSE;
    SNGDIM = $TRUE;
    CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
            IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
            OUT (RADSPTR);

    IF (RADSPTR NE $NULL)
        THEN PERFORM resolution_advisories_store_in_pair_record;
        ELSE PREC.POSCMD = $ONEHIT;
            PREC.PIFR = $FALSE;
            IF (ELENTRY.DELREQ EQ $FALSE)
                THEN ELENTRY.DELREQ = $TRUE;
                ELSE;

END initial_resolution_advisory_selection;
```

```
----------------------------------------------------------------------------------------
PROCESS previous_resolution_advisory_modification_tests;

     <Resolution advisories were selected on a previous scan.  Determine if
      they should be recalculated for any reason.>

     CLEAR flag indicating resolution advisories have been recalculated;
     PERFORM previous_resolution_advisories_recalculation_checks;
     IF (resolution advisories were not recalculated)
          THEN IF ((there is a controlled AC in the subject pair) AND
                         ((the controlled AC has not yet been given a resolution
                         advisory) AND
                         ((detection logic determined that the controlled AC
                         should receive a resolution advisory) OR
                         (number of AC in conflict table GT single pair))))
                    THEN PERFORM resolution_advisory_addition_for_controlled_
                                                                    aircraft;
                    ELSE PERFORM positive_negative_resolution_advisory_transition_
                                                                        test;
                         IF (resolution advisories have not been recalculated)
                              THEN PERFORM previous_resolution_advisories_
                                                          non_response_test;
                              ELSE;
               IF (resolution advisories were not recalculated)
                    THEN save horizontal and vertical miss distances from the
                              encounter list entry in the pair record;
                    ELSE;
          ELSE;

END previous_resolution_advisory_modification_tests;
```

```
---------------------------------------------------------------------------

PROCESS previous_resolution_advisory_modification_tests;


     RECALC = $FALSE;

     PERFORM previous_resolution_advisories_recalculation_checks;

     IF (RECALC EQ $FALSE)

          THEN IF (((ACID1.CONC EQ $TRUE) OR (ACID2.CONC EQ $TRUE)) AND

                         (PREC.PIFR EQ $FALSE) AND ((ELENTRY.IFRFLG EQ $TRUE) OR

                         (ACID1.CTPTR.NAC GT APAIR)))

                    THEN PERFORM resolution_advisory_addition_for_controlled_

                                                                    aircraft;

                    ELSE PERFORM positive_negative_resolution_advisory_transition_

                                                                        test;

                         IF (RECALC EQ $FALSE)

                              THEN PERFORM previous_resolution_advisories_

                                                            non_response_test;

                              ELSE;

               IF (RECALC EQ $FALSE)

                    THEN PREC.PMD = ELENTRY.MD2;

                         PREC.PVMD = ELENTRY.ALT;

                    ELSE;

          ELSE;

END previous_resolution_advisory_modification_tests;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_posting_from_pair_record_to_conflict_table;

     <Select effective resolution advisory for conflict table entry from
      all pair records associated with the subject aircraft.>

     LOOP:
          Select conflict table entry of next aircraft of subject pair;
     EXITIF (both aircraft in pair are done);
          CALL VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
          CALL HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
     ENDLOOP;

END resolution_advisory_posting_from_pair_record_to_conflict_table;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_posting_from_pair_record_to_conflict_table;

    LOOP;
        Select CTENTRY of next aircraft of subject pair;
    EXITIF (both aircraft in pair are done);
        CALL VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
                IN (TACID, PREC)
                INOUT (TACID.CTE.VMAN, TACID.CTE.ACIDV, TACID.CTE.MULTV);
        CALL HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
                IN (TACID, PREC)
                INOUT (TACID.CTE.HMAN, TACID.CTE.ACIDH, TACID.CTE.MULTH);
    ENDLOOP;

END resolution_advisory_posting_from_pair_record_to_conflict_table;
```

```
--------------------------------------------------------------------------------
PROCESS other_ATARS_sites_resolution_advisories_determination;

     <Determine what advisories (if any) are being given to the subject
      aircraft from other, higher priority sites.>

     CLEAR temporary storage for resolution advisories from other ATARS sites;
     LOOP;
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records);
          IF (this pair record has a resolution advisory for DABS AC of subject
                    pair AND is from a higher priority non-connected site)
               THEN save more severe of this resolution advisory and resolution
                         advisory already saved in the horizontal and vertical
                         dimensions from other non-connected sites;
               ELSE;
     ENDLOOP;

END other_ATARS_sites_resolution_advisories_determination;
```

```
-------------------------------------------------------------------------------------

PROCESS other_ATARS_sites_resolution_advisories_determination;

     OSHMAN = $NULLRES;
     OSVMAN = $NULLRES;
     LOOP:
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records);
          IF ((TPREC.ATSID GT SYSTEM.OWNID) AND (site is non-connected)
                    AND (TPREC.ac1.PAC EQ PREC.ac1.PAC))
               THEN OSHMAN = EFFHRA(TPREC.ac1.PHMAN,OSHMAN);
                    OSVMAN = EFFVRA(TPREC.ac1.PVMAN,OSVMAN);
               ELSE;
     ENDLOOP;

END other_ATARS_sites_resolution_advisories_determination;
```

```
--------------------------------------------------------------------------------
PROCESS positive_negative_resolution_advisory_horizontal_transition_test;

     <Check if horizontal resolution advisories may transition between
      positive and negative.>

     IF (either AC is detected to be turning)
          THEN SET negative horizontal resolution advisory threshold to modified
                    value;
          ELSE SET negative horizontal resolution advisory threshold to default
                    value;
     IF (current horizontal miss distance LT negative horizontal resolution
               advisory threshold)
          THEN IF (pair record has negative horizontal resolution advisories)
                    THEN indicate that transition is appropriate;
                    ELSE;

          ELSE IF (pair record has positive horizontal resolution advisories)
                    THEN indicate that transition is appropriate;
                    ELSE;

END positive_negative_resolution_advisory_horizontal_transition_test;
```

```
--------------------------------------------------------------------------------
PROCESS positive_negative_resolution_advisory_horizontal_transition_test;

    IF (((ACID1.TURN NE $STRAIGHT) AND (ACID1.TURN NE $HUHMINUS) AND
            (ACID1.TURN NE $HUHPLUS)) OR ((ACID2.TURN NE $STRAIGHT) AND
            (ACID2.TURN NE $HUHMINUS) AND (ACID2.TURN NE $HUHPLUS)))
        THEN MDTHM = RESADV.MDTHMSQ;
        ELSE MDTHM = RESADV.MDTHSQ;

    IF (ELENTRY.MD2 LT MDTHM)
        THEN IF ((PREC.POSCMD EQ $NEG) AND (PREC.ac1.PHMAN NE $NULLRES))
                THEN RECALC = $TRUE;
                ELSE;

        ELSE IF (((PREC.POSCMD EQ $POS) OR (PREC.POSCMD EQ $DOUBLE)) AND
                    (PREC.ac1.PHMAN NE $NULLRES))
                THEN RECALC = $TRUE;
                ELSE;

END positive_negative_resolution_advisory_horizontal_transition_test;
```

```
------------------------------------------------------------------------------
PROCESS positive_negative_resolution_advisory_transition_test;

     <Check if resolution advisories in either dimension may transition
      between positive and negative.>

     IF ((conflict control variable is set for negative resolution advisories) OR
               (last positive resolution advisory has been displayed in the
                aircraft long enough that it may be changed))
          THEN IF (pair record has horizontal resolution advisories)
                    THEN PERFORM positive_negative_resolution_advisory_horizontal_
                                                                  transition_test;
                    ELSE;
               IF (transition is not yet possible)
                    THEN IF (pair record has vertical resolution advisories)
                              THEN PERFORM positive_negative_resolution_
                                             advisory_vertical_transition_test;
                              ELSE;
                    ELSE;
               IF (transition can be attempted)
                    THEN SET flag to indicate single dimension resolution advisories
                              are preferred;
                         CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;
                         IF (resolution advisories selected for the pair)
                              THEN PERFORM
                                        resolution_advisories_store_in_pair_record;
                              ELSE IF (this pair flagged for normal resolution)
                                        THEN flag this pair for delayed
                                                  resolution;
                                        ELSE save the horizontal and vertical
                                                  miss distances from the
                                                  encounter list entry in the
                                                  pair record;
                    ELSE;
          ELSE;
END positive_negative_resolution_advisory_transition_test;
```

```
--------------------------------------------------------------------------------------------
PROCESS positive_negative_resolution_advisory_transition_test;

     IF ((PREC.POSCMD EQ $NEG) OR ((PREC.TSTART + TSCMD) LT SYSVAR.CTIME))
          THEN IF (PREC.ac1.PHMAN NE $NULLRES)
                    THEN PERFORM positive_negative_resolution_advisory_horizontal_
                                                                transition_test;
                    ELSE;
               IF (RECALC EQ $FALSE)
                    THEN IF (PREC.ac1.PVMAN NE $NULLRES)
                              THEN PERFORM positive_negative_resolution_advisory_
                                                  horizontal_transition_test;
                              ELSE;
                    ELSE;
               IF (RECALC EQ $TRUE)
                    THEN SNGDIM = $TRUE;
                         CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
                                IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
                                OUT (RADSPTR);
                         IF (RADSPTR NE $NULL)
                              THEN PERFORM
                                        resolution_advisories_store_in_pair_record;
                              ELSE IF (ELENTRY.DELREQ EQ $FALSE)
                                        THEN ELENTRY.DELREQ = $TRUE;
                                        ELSE PREC.PMD = ELENTRY.MD2;
                                             PREC.PVMD = ELENTRY.ALT;
                    ELSE;
          ELSE;

END positive_negative_resolution_advisory_transition_test;
```

```
PROCESS positive_negative_resolution_advisory_vertical_transition_test;

     <Check if vertical resolution advisories may transition between positive
      and negative or among the various negatives (negative and VSL).>

     IF (current altitude separation LT negative vertical resolution advisory
              threshold)
         THEN IF ((pair record has negative vertical resolution advisories) AND
                        (current altitude separation LT percentage of negative
                        resolution advisory threshold))
                   THEN SET flag indicating resolution advisories should be
                             recalculated;
                   ELSE;
         ELSE IF (pair record has positive vertical resolution advisories)
                   THEN IF (vertical tau is negative)
                             THEN SET flag indicating resolution advisories
                                       should be recalculated;
                             ELSE;
                   ELSE save the resolution advisories from the pair record
                             into a RADS for the call to the VSL logic;
                        SET negative flag in the RADS;
                        SET flags for single dimension vertical resolution
                             advisories;
                        SET flag to indicate which AC are maneuvered;
                        CALL VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION;
                        IF (vertical resolution advisories in RADS NE
                                  vertical resolution advisories in pair record)
                             THEN PERFORM resolution_advisories_store_in_pair_
                                                                           record;
                             ELSE;

END positive_negative_resolution_advisory_vertical_transition_test;
```

```
--------------------------------------------------------------------------------

PROCESS positive_negative_resolution_advisory_vertical_transition_test;


     IF (ELENTRY.ALT LT ASEP)
          THEN IF ((PREC.POSCMD EQ $NEG) AND (PREC.ac1.PVMAN NE $NULLRES)
                         AND (ELENTRY.ALT LT (ATBZP * ASEP)))
                    THEN RECALC = $TRUE;
                    ELSE;
          ELSE IF (((PREC.POSCMD EQ $DOUBLE) OR (PREC.POSCMD EQ $POS)) AND
                         (PREC.ac1.PVMAN NE $NULLRES))
                    THEN IF ((ELENTRY.TV LT 0)
                              THEN RECALC = $TRUE;
                              ELSE;
                    ELSE RADS.NEGATIVE = $TRUE;
                         RADS.VERT = $TRUE;
                         RADS.SINGLE = $TRUE;
                         RADS.V1 = PREC.ac1.PVMAN;
                         RADS.V2 = PREC.ac2.PVMAN;
                         RADS.H1 = $NULLRES;
                         RADS.H2 = $NULLRES;
                         IF ((PREC.ac1.PVMAN NE $NORES) AND
                                   (PREC.ac2.PVMAN NE $NORES))
                              THEN RADS.CMDED_CMDED = $TRUE;
                         ELSEIF (PREC.ac1.PVMAN NE $NORES)
                              THEN RADS.CMDED_UNCMDED = $TRUE;
                         OTHERWISE RADS.UNCMDED_CMDED = $TRUE;
                         CALL VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION
                                IN (RADS, ACID1, ACID2, PREC)
                                OUT (RADS.V1, RADS.V2);
                         IF ((RADS.V1 NE PREC.ac1.PVMAN) OR
                                   (RADS.V2 NE PREC.ac2.PVMAN))
                              THEN PERFORM resolution_advisories_store_in_
                                                                       pair_record;
                              ELSE;

END positive_negative_resolution_advisory_vertical_transition_test;
----------------------- MASTER RESOLUTION LOW-LEVEL LOGIC -------------------------
```

```
------------------------------------------------------------------------------------

PROCESS previous_resolution_advisories_non_response_test;

    <Check for aircraft non-response to previous positive single dimension
     resolution advisories.>

    IF ((conflict control variable indicates that positive single dimension
              resolution advisories have been given) AND (enough time has elapsed
              that response to the resolution advisories should have been detected
              in the form of increasing resolution dimension miss distance))
        THEN IF (pair record contains positive horizontal resolution advisories)
                  THEN IF (current scan's horizontal miss distance LT previous
                                 scan's horizontal miss distance)
                            THEN SET flag to indicate that resolution
                                      advisories should be recalculated;
                            ELSE;
                  ELSE IF (current scan's vertical miss distance LT previous scan's
                                 vertical miss distance)
                            THEN SET flag to indicate that resolution
                                      advisories should be recalculated;
                            ELSE;
              IF (new resolution advisories should be selected)
                  THEN SET flag to prefer double dimension resolution advisories;
                       CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;
                       IF (resolution advisories were selected for the pair)
                            THEN PERFORM resolution_advisories_store_in_pair_
                                                                        record;
                            ELSE IF (this pair was flagged for normal resolution)
                                      THEN flag this pair for delayed resolution;
                                      ELSE save current scan's horizontal and
                                                vertical miss distances from
                                                encounter list in pair record;
                  ELSE;
        ELSE;

END previous_resolution_advisories_non_response_test;
```

```
--------------------------------------------------------------------------------

PROCESS previous_resolution_advisories_non_response_test;

    IF ((PREC.POSCMD EQ $POS) AND ((PREC.TSTART + TRECOM) GT SYSVAR.CTIME))
        THEN IF (PREC.ac1.PHMAN NE $NULLRES)
                THEN IF (ELENTRY.MD2 LT PREC.PMD)
                        THEN RECALC = $TRUE;
                        ELSE;
                ELSE IF (ELENTRY.ALT LT PREC.PVMD)
                        THEN RECALC = $TRUE;
                        ELSE;
            IF (RECALC EQ $TRUE)
                THEN SNGDIM = $FALSE;
                    CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
                        IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
                        OUT (RADSPTR);
                    IF (RADSPTR NE $NULL)
                        THEN PERFORM resolution_advisories_store_in_pair_
                                                                    record;
                        ELSE IF (ELENTRY.DELREQ EQ $FALSE)
                                THEN ELENTRY.DELREQ = $TRUE;
                                ELSE PREC.PMD = ELENTRY.MD2;
                                     PREC.PVMD = ELENTRY.ALT;
                ELSE;
        ELSE;

END previous_resolution_advisories_non_response_test;
```

```
--------------------------------------------------------------------------------
PROCESS previous_resolution_advisories_recalculation_checks;

     <Determine if resolution advisories should be recalculated because of
      incompatibility with resolution advisories from other sites or BCAS
      that were selected on the same scan or because the aircraft
      characteristics have changed significantly within two scans of
      resolution advisory selection.>

     IF (conflict control variable indicates that resolution advisories were
               incompatible and should be recalculated)
          THEN indicate whether single or double dimension resolution advisories
                    are desired based on conflict control variable;
               SET flag to indicate resolution advisories were recalculated;
               CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;
          ELSE PERFORM PSEP_model_validation;
               IF (resolution advisories must be recalculated because of PSEP model
                         validation logic)
                    THEN indicate double dimension resolution advisories preferred;
                         CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;

     IF (attempted to recalculate resolution advisries)
          THEN IF (resolution advisories are selected for the pair)
                    THEN PERFORM resolution_advisories_store_in_pair_record;
                    ELSE IF (pair flagged for normal resolution)
                              THEN flag this pair for delayed resolution;
                              ELSE save horizontal and vertical miss distance from
                                        encounter list in pair record;
          ELSE;

END previous_resolution_advisories_recalculation_checks;
```

```
--------------------------------------------------------------------------------
PROCESS previous_resolution_advisories_recalculation_checks;

    IF ((PREC.POSCMD EQ $RCMSNG) OR (PREC.POSCMD EQ $RCMDBL))
        THEN IF (PREC.POSCMD EQ $RCMSNG)
                THEN SNGDIM = $TRUE;
                ELSE SNGDIM = $FALSE;
             RECALC = $TRUE;
             CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
                    IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
                    OUT (RADSPTR);
        ELSE PERFORM PSEP_model_validation;
             IF (RECALC EQ $TRUE)
                THEN SNGDIM = $FALSE;
                    CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
                           IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
                           OUT (RADSPTR);
    IF (RECALC EQ $TRUE)
        THEN IF (RADSPTR NE $NULL)
                THEN PERFORM resolution_advisories_store_in_pair_record;
                ELSE IF (ELENTRY.DELREQ EQ $FALSE)
                        THEN ELENTRY.DELREQ = $TRUE;
                        ELSE PREC.PMD = ELENTRY.MD2;
                             PREC.PVMD = ELENTRY.ALT;
        ELSE;

END previous_resolution_advisories_recalculation_checks;
```

```
--------------------------------------------------------------------------------
PROCESS PSEP_model_validation;

     <Determine if the conditions that existed when resolution advisories
      were selected have changed.  If they have changed detrimentally in
      the resolution dimension, then reselect resolution advisories.>

     IF ((resolution advisories were selected recently enough in the pair record)
              AND (model validation logic was not the last
              cause of selecting resolution advisories) AND
              (single dimension resolution advisories are in the pair record))
          THEN IF (there are horizontal resolution advisories in the pair record)
                    THEN IF (either AC's turn state is detrimental to previous
                                   horizontal resolution advisory and previous
                                   turn state for that AC)
                              THEN SET flag indicating resolution advisories
                                        should be recalculated;
                              ELSE;
                    ELSE IF (vertical velocity difference between two aircraft is
                                   different from when resolution advisories were
                                   selected AND difference between vertical
                                   velocity differences is detrimental)
                              THEN SET flag indicating resolution advisories
                                        should be recalculated;
                              ELSE;
          ELSE;

END PSEP_model_validation;
```

```
--------------------------------------------------------------------------------

PROCESS PSEP_model_validation;

    IF ((SYSVAR.CTIME LT (PREC.MVWAIT + (TVALID * SYSTEM.SCAN))) AND
            (PREC.MVDONE EQ $FALSE) AND
            ((PREC.POSCMD EQ $NEG) OR (PREC.POSCMD EQ $POS)))
        THEN IF (PREC.ac1.PHMAN NE $NULLRES)
            THEN IF ((DETRIMH(PREC.ac1.PHMAN,ACID1.TURN,PREC.ac1.MVT) EQ
                        $TRUE) OR (DETRIMH(PREC.ac2.PHMAN,ACID2.TURN,
                        PREC.ac2.MVT) EQ $TRUE))
                    THEN RECALC = $TRUE;
                    ELSE;
            ELSE IF (DETRIMV(PREC.ac1.PVMAN,PREC.ac2.PVMAN,
                        ((ACID2.ZD - ACID1.ZD) - PREC.MVVRZ)) EQ $TRUE)
                    THEN RECALC = $TRUE;
                    ELSE;
        ELSE;

END PSEP_model_validation;
```

```
--------------------------------------------------------------------------------------

PROCESS resolution_advisories_store_in_pair_record;

     <Resolution advisories were selected this scan by RAER.  If any of the
      resolution advisories are not exactly the same as those that
      currently exist in the pair record, save the new advisories in the
      pair record and set the timer, SEND flags, and POSCMD variable.>

     Save the horizontal and vertical miss distances from the encounter list in
          the pair record;
     IF (selected resolution advisories are not exactly the same as those in the
               pair record)
          THEN save the selected resolution advisories in the pair record;
               SET the timer in pair record to the current time;
               IF (both AC are maneuvered)
                    THEN SET send flag for both AC;
               ELSEIF (first AC maneuvered)
                    THEN SET send flag for first AC;
               OTHERWISE SET send flag for second AC;

               IF (there are negative resolution advisories selected)
                    THEN set the conflict control variable to indicate negative
                              resolution advisories selected;
               ELSEIF (positive single dimension resolution advisories in the pair
                         record)
                    THEN set the conflict control variable to indicate positive
                              single dimension advisories selected;
               OTHERWISE SET the conflict control variable to indicate double
                              dimension resolution advisories selected;

               SET flag indicating resolution advisories selected this scan;

          ELSE;   <nothing needs to be done.>

END resolution_advisories_store_in_pair_record;
```

```
--------------------------------------------------------------------------------

PROCESS resolution_advisories_store_in_pair_record;

     PREC.PMD = ELENTRY.MD2;
     PREC.PVMD = ELENTRY.ALT;

     IF ((RADSPTR.H1 NE PREC.ac1.PHMAN) OR (RADSPTR.H2 NE PREC.ac2.PHMAN)
               OR (RADSPTR.V1 NE PREC.ac1.PVMAN) OR (RADSPTR.V2 NE PREC.ac2.PVMAN))
          THEN PREC.ac1.PHMAN = RADSPTR.H1;
               PREC.ac2.PHMAN = RADSPTR.H2;
               PREC.ac1.PVMAN = RADSPTR.V1;
               PREC.ac2.PVMAN = RADSPTR.V2;
               PREC.TSTART = SYSVAR.CTIME;

               IF (RADSPTR.CMDED_CMDED EQ $TRUE)
                    THEN PREC.ac1.SEND = $TRUE;
                         PREC.ac2.SEND = $TRUE;
               ELSEIF (RADSPTR.CMDED_UNCMDED EQ $TRUE)
                    THEN PREC.ac1.SEND = $TRUE;
               OTHERWISE PREC.ac2.SEND = $TRUE;

               IF (RADSPTR.NEGATIVE EQ $TRUE)
                    THEN PREC.POSCMD = $NEG;
               ELSEIF (RADSPTR.SINGLE = $TRUE)
                    THEN PREC.POSCMD = $POS;
               OTHERWISE PREC.POSCMD = $DOUBLE;


               RASELECT = $TRUE;
          ELSE;   <nothing needs to be done.>

END resolution_advisories_store_in_pair_record;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_addition_for_controlled_aircraft;

     <This conflict pair was previously being resolved by maneuvering only the
      uncontrolled AC.  The detection logic has now determined that the conflict
      has reached the point where the controlled AC must also be maneuvered
      (or the pair has now become part of a multi-AC conflict, in which case
      Master Resolution has made the decision to maneuver the controlled AC).
      Call RAER to select advisories for both AC.>

     SET flag indicating resolution advisories have been recalculated;
     SET flag in pair record indicating that controlled AC should receive
         a resolution advisory;
     SET flag to indicate single dimension resolution advisories preferred;
     CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;

     IF (resolution advisories selected for the pair)
         THEN PERFORM resolution_advisories_store_in_pair_record;
         ELSE CLEAR flag in pair record to indicate that resolution advisories for
                   controlled aircraft is selected;
              IF (this pair is flagged for normal resolution)
                   THEN flag this pair for delayed resolution;
                   ELSE save the computed horizontal and vertical miss
                             distances from the encounter list in the pair
                             record;

END resolution_advisory_addition_for_controlled_aircraft;
```

```
--------------------------------------------------------------------------------------

PROCESS resolution_advisory_addition_for_controlled_aircraft;

     RECALC = $TRUE;
     PREC.PIFR = $TRUE;
     SNGDIM = $TRUE;
     CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
             IN (ELENTRY, PREC, ASEP, SNGDIM, MRNCAP)
             OUT (RADSPTR);

     IF (RADSPTR NE $NULL)
          THEN PERFORM resolution_advisories_store_in_pair_record;
          ELSE PREC.PIFR = $FALSE;
               IF (ELENTRY.DELREQ EQ $FALSE)
                    THEN ELENTRY.DELREQ = $TRUE;
                    ELSE PREC.PMD = ELENTRY.MD2;
                         PREC.PVMD = ELENTRY.ALT;

END resolution_advisory_addition_for_controlled_aircraft;
```

```
--------------------------------------------------------------------------------
ROUTINE ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION
   IN (aircraft state vectors)
   OUT (altitude separation threshold parameter):

     <Determine the altitude separation threshold used for positive/negative
        resolution advisory selection and transition.>

     IF (either aircraft above floor of ultra-high airspace)
          THEN SET altitude separation threshold to ultra-high altitude
                    positive advisory threshold;
     ELSEIF (either aircraft above floor of positive controlled airspace)
          THEN SET altitude separation threshold to high altitude positive
                    advisory threshold;

     ELSEIF (either aircraft is controlled)
          THEN SET altitude separation threshold to low altitude positive
                    advisory threshold for uncontrolled/controlled
                    or controlled/controlled conflict pairs;
     OTHERWISE SET altitude separation threshold to low altitude positive
                    advisory threshold for uncontrolled/uncontrolled
                    aircraft;

END ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION;
```

```
------------------------------------------------------------------------------

ROUTINE ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION
   IN (ACID1, ACID2)
   OUT (ASEP);

     IF ((ACID1.Z GT ALUH) OR (ACID2.Z GT ALUH))
          THEN ASEP = ASEPU;
     ELSEIF ((ACID1.Z GT ALPC) OR (ACID2.Z GT ALPC))
          THEN ASEP = ASEPH;
     ELSEIF ((ACID1.CUNC EQ $TRUE) OR (ACID2.CUNC EQ $TRUE))
          THEN ASEP = ASEPIL;
     OTHERWISE ASEP = ASEPL;

END ALTITUDE_SEPARATION_THRESHOLD_DETERMINATION;
```

```
-------------------------------------------------------------------------

ROUTINE CONFLICT_CONTROL_VARIABLE_UPDATE
   IN (maneuvering target threat and resolution advisory flags)
   INOUT (conflict control variable);

     <The conflict control variable (POSCMD) is used to implement the
      2-out-of-3 rule for detecting the need for resolution advisories
      before actually selecting them.  It is also used to indicate the
      severity of the advisories in the pair record (negative, positive or
      double dimension positive).  Another function of POSCMD is to pass
      information from RAR Processing Task to Master Resolution if
      the previously selected resolution advisories were incompatible
      with those from another source.  This process is used only to
      update POSCMD for the 2-out-of-3 rule.>

     IF (maneuvering target threat flag is set)
         THEN conflict control variable should be set to indicate resolution
                   advisories are necessary;
         ELSE IF (resolution advisory flag is set)
                   THEN IF (conflict control variable shows that the pair record
                                 was created this scan)
                             THEN SET conflict control variable for one hit;
                             ELSE SET conflict control variable to indicate
                                       resolution advisoress are necessary;
                   ELSE IF (conflict control variable shows pair record created
                                 this scan OR one hit recorded)
                             THEN SET conflict control vari[illegible] one miss;
                             ELSE SET conflict control var[illegible]e no
                                       resolution advisory necessary;

END CONFLICT_CONTROL_VARIABLE_UPDATE;
```

```
-------------------------------------------------------------------------------------------
ROUTINE CONFLICT_CONTROL_VARIABLE_UPDATE
   IN (ELENTRY.CMDFLG, ELENTRY.NTTFLG)
   INOUT (PREC.POSCMD);

     IF (ELENTRY.NTTFLG EQ $TRUE)
          THEN PREC.POSCMD = $RANEC;
          ELSE IF (ELENTRY.CMDFLG = $TRUE)
                    THEN IF (PREC.POSCMD EQ $NOTSET)
                              THEN PREC.POSCMD = $ONEHIT;
                              ELSE PREC.POSCMD = $RANEC;
                    ELSE IF ((PREC.POSCMD EQ $ONEHIT) OR (PREC.POSCMD EQ $NOTSET))
                              THEN PREC.POSCMD = $ONEMIS;
                              ELSE PREC.POSCMD = $NORA;

END CONFLICT_CONTROL_VARIABLE_UPDATE;
```

```
--------------------------------------------------------------------------------
ROUTINE HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
   IN (AC state vector, pair record)
   OUT (horizontal maneuver, pair record pointer and multiplicity count in
        conflict table);

     <Resolution advisories have been selected by RAER on this scan.  This
      routine determines what (if any) horizontal resolution advisories
      should appear in the conflict table entry.>

     IF (horizontal resolution advisories in the pair record)
         THEN PERFORM horizontal_resolution_advisory_in_pair_record;
         ELSE PERFORM horizontal_resolution_advisory_not_in_pair_record;

END HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
```

```
---------------------------------------------------------------------------------

ROUTINE HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
   IN (ACID, PREC)
   OUT (CTENTRY.HMAN, CTENTRY.ACIDH, CTENTRY.HULTH);

     IF (PREC.ac.PHMAN NE $NULLRES)
          THEN PERFORM horizontal_resolution_advisory_in_pair_record;
          ELSE PERFORM horizontal_resolution_advisory_not_in_pair_record;

END HORIZONTAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
```

```
--------------------------------------------------------------------------------------
PROCESS horizontal_resolution_advisory_in_pair_record;

     <A horizontal resolution advisory is in the pair record for this AC.
      This process determines the effective horizontal resolution
      advisory for the conflict table entry based on this and all other
      pair records with horizontal advisories for this AC.>

     IF (more than one pair record causing a horizontal resolution advisory in the
               conflict table entry to this AC)
          THEN PERFORM horizontal_resolution_advisory_selection;
     ELSEIF ((one pair record causing a horizontal resolution advisory to this AC
                    in the conflict table entry) AND (the conflict table entry points
                    to this pair record))
          THEN SET horizontal resolution advisory in conflict table entry to the
                    resolution advisory in the pair record;
     ELSEIF (one pair record causing a horizontal resolution advisory in the
               conflict table entry)
          THEN place effective resolution advisory in the conflict table entry;
               Increment number of pair records causing a horizontal resolution
                    advisory to this AC;
     OTHERWISE SET resolution advisory in conflict table entry to resolution
                    advisory in pair record;
               SET number of pair records causing horizontal resolution
                    advisory to one;
               SET conflict table entry pair record pointer to point to this
                    pair record;

END horizontal_resolution_advisory_in_pair_record;
```

```
---------------------------------------------------------------------------------------

PROCESS horizontal_resolution_advisory_in_pair_record;

     IF (CTENTRY.MULTH GT 1)
          THEN PERFORM horizontal_resolution_advisory_selection;
     ELSEIF ((CTENTRY.MULTH EQ 1) AND (CTENTRY.ACIDH EQ PREC))
          THEN CTENTRY.HMAN = PREC.ac.PHMAN;
     ELSEIF (CTENTRY.MULTH EQ 1)
          THEN CTENTRY.HMAN = EFFHRA(PREC.ac.PHMAN,CTENTRY.HMAN);
               CTENTRY.MULTH = ITWO;
     OTHERWISE CTENTRY.HMAN = PREC.ac.PHMAN;
               CTENTRY.MULTH = 1;
               CTENTRY.ACIDH = PREC;

END horizontal_resolution_advisory_in_pair_record;
```

```
--------------------------------------------------------------------------------
PROCESS horizontal_resolution_advisory_not_in_pair_record;

    <There is not a horizontal resolution advisory for this AC in the pair
     record.  This process determines if a horizontal resolution advisory
     may have been in the pair record on the previous scan.  If so,
     the effective horizontal resolution advisory in the conflict table
     entry should be recomputed.

     Multiplicity is the number of pair records contributing to the
     resolution advisory in one dimension for an aircraft (MULTH).>

    IF (horizontal resolution advisory multiplicity in conflict table GT one)
         THEN PERFORM horizontal_resolution_advisory_selection;
    ELSEIF ((horizontal resolution advisory multiplicity in conflict table entry
                   EQ one) AND (conflict table entry points to this pair record))
         THEN SET horizontal resolution advisory in conflict table entry to null;
              SET horizontal resolution advisory multiplicity to zero;
              SET pair record pointer in conflict table entry to null;
    OTHERWISE;
              <no horizontal resolution advisory in conflict table entry, or only
               one other pair record causing a horizontal resolution advisory.>

END horizontal_resolution_advisory_not_in_pair_record;
```

```
-----------------------------------------------------------------------------------------
PROCESS horizontal_resolution_advisory_not_in_pair_record;


     IF (CTENTRY.NULTH GT 1)
          THEN PERFORM horizontal_resolution_advisory_selection;
     ELSEIF ((CTENTRY.NULTH EQ 1) AND (CTENTRY.ACIDH EQ PREC))
          THEN CTENTRY.HMAN = $NULLRES;
               CTENTRY.NULTH = 0;
               CTENTRY.ACIDH = $NULL;
     OTHERWISE:
                <no horizontal RA in conflict table entry, or only
                 one other pair record causing a horizontal RA.>

END horizontal_resolution_advisory_not_in_pair_record;
```

```
--------------------------------------------------------------------------------
PROCESS horizontal_resolution_advisory_selection;

     <This process examines all pair records in which the subject AC is
      contained.  The effective horizontal resolution advisory is selected
      from all of the pair records and placed in the conflict table entry.>

     SET horizontal resolution advisory in conflict table entry to null;
     SET conflict table entry horizontal pair record pointer to null;
     SET horizontal resolution advisory multiplicity count to zero;

     LOOP:
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records);
          IF (subject AC in this pair record)
               THEN IF (horizontal resolution advisory in this pair record is
                              not null)
                         THEN place effective horizontal resolution advisory
                                   in conflict table entry;
                              Increment horizontal resolution advisory multiplicity
                                   count;
                              IF (conflict table entry pair record pointer EQ null)
                                   THEN SET conflict table entry pair record pointer
                                             to point to this pair record;
                                   ELSE;
                         ELSE;
               ELSE;
     ENDLOOP;

END horizontal_resolution_advisory_selection
```

```
--------------------------------------------------------------------------------

PROCESS horizontal_resolution_advisory_selection;

     CTENTRY.HMAN = $NULLRES;
     CTENTRY.ACIDH = $NULL;
     CTENTRY.MULTH = 0;
     LOOP;
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records);
          IF (ACID.CTENTRY EQ TPREC.ac1.PAC) OR (ACID.CTENTRY EQ TPREC.ac2.PAC))
               THEN IF (TPREC.ac.PHMAN NE $NULLRES)
                         THEN CTENTRY.HMAN = EFFHRA(TPREC.ac.PHMAN,CTENTRY.HMAN);
                              CTENTRY.MULTH = CTENTRY.MULTH + 1;
                              IF (CTENTRY.ACIDH EQ $NULL)
                                   THEN CTENTRY.ACIDH = TPREC;
                                   ELSE;
                         ELSE;
               ELSE;
     ENDLOOP;

END horizontal_resolution_advisory_selection
```

```
--------------------------------------------------------------------------------

ROUTINE VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
   IN (AC state vector, pair record)
   OUT (conflict table entry: vertical maneuver, pair record pointer,
          multiplicity);

     <Resolution advisories have been selected by RAER on this scan.  This
      routine determines what (if any) vertical resolution advisories
      should appear in the conflict table entry.>

     IF (pair record has vertical resolution advisories)
          THEN PERFORM vertical_resolution_advisory_in_pair_record;
          ELSE PERFORM vertical_resolution_advisory_not_in_pair_record;

END VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
```

```
--------------------------------------------------------------------------------

ROUTINE VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE
   IN (ACID, PREC)
   OUT (CTENTRY.VMAN, CTENTRY.ACIDV, CTENTRY.MULTV);

     IF (PREC.ac.PVMAN NE $NULLRES)
          THEN PERFORM vertical_resolution_advisory_in_pair_record;
          ELSE PERFORM vertical_resolution_advisory_not_in_pair_record;

END VERTICAL_RESOLUTION_ADVISORY_POSTING_TO_CONFLICT_TABLE;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_resolution_advisory_in_pair_record;

     <A vertical resolution advisory is in the pair record for this AC.
      This process determines the effective vertical resolution
      advisory for the conflict table entry based on this and all other
      pair records with vertical advisories for this AC.>

     IF (more than one pair record causing a vertical resolution advisory in the
               conflict table entry to this AC)
          THEN PERFORM vertical_resolution_advisory_selection;
     ELSEIF (one pair record causing a vertical resolution advisory to this AC
               in the conflict table entry AND the conflict table entry points
               to this pair record)
          THEN SET vertical resolution advisory in conflict table entry to the
                    vertical resolution advisory in the pair record;
     ELSEIF (one pair record causing a vertical resolution advisory in the conflict
               table entry)
          THEN save effective vertical resolution advisory in the conflict
                    table entry;
               Increment number of pair records causing a vertical resolution
                    advisory to this AC;

     OTHERWISE SET resolution advisory in conflict table entry to resolution
                    advisory in pair record;
               SET number of pair records causing vertical resolution advisory
                    to this AC to one;
               SET conflict table entry pair record pointer to point to this
                    pair record;

END vertical_resolution_advisory_in_pair_record;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_resolution_advisory_in_pair_record;

     IF (CTENTRY.MULTV GT 1)
          THEN PERFORM vertical_resolution_advisory_selection;
     ELSEIF ((CTENTRY.MULTV EQ 1) AND (CTENTRY.ACIDV EQ PREC))
          THEN CTENTRY.VMAN = PREC.ac.PVMAN;
     ELSEIF (CTENTRY.MULTV EQ 1)
          THEN CTENTRY.VMAN = EFFVRA(PREC.ac.PHMAN,CTENTRY.VMAN);
               CTENTRY.MULTV = ITWO;
     OTHERWISE CTENTRY.VMAN = PREC.ac.PVMAN;
               CTENTRY.MULTV = 1;
               CTENTRY.ACIDV = PREC;

END vertical_resolution_advisory_in_pair_record;
```

```
------------------------------------------------------------------------------------
PROCESS vertical_resolution_advisory_not_in_pair_record;

     <There is not a vertical resolution advisory for this AC in the pair
      record.  This process determines if a vertical resolution advisory
      may have been in the pair record on the previous scan.  If so,
      the effective vertical resolution advisory in the conflict table
      entry should be recomputed.

      Multiplicity is the number of pair records contributing to the
      resolution advisory in one dimension for an aircraft (MULTV).>

     IF (vertical resolution advisory multiplicity in conflict table entry GT one)
          THEN PERFORM vertical_resolution_advisory_selection;
     ELSEIF ((vertical resolution advisory multiplicity in conflict table entry
               EQ one) AND (conflict table entry points to this pair record))
          THEN SET vertical resolution advisory in conflict table entry to null;
               SET vertical resolution advisory multiplicity to zero;
               SET pair record pointer in conflict table entry to null;
     OTHERWISE:   <no vertical resolution advisory in conflict table entry, or only
                    one other pair record causing a vertical resolution advisory.>

END vertical_resolution_advisory_not_in_pair_record;
```

```
---------------------------------------------------------------------------------------

PROCESS vertical_resolution_advisory_not_in_pair_record;

    IF (CTENTRY.MULTV GT 1)
        THEN PERFORM vertical_resolution_advisory_selection;
    ELSEIF ((CTENTRY.MULTV EQ 1) AND (CTENTRY.ACIDV EQ PREC))
        THEN CTENTRY.VMAN = $NULLRES;
             CTENTRY.MULTV = 0;
             CTENTRY.ACIDV = $NULL;
    OTHERWISE:   <no vertical RA in conflict table entry, or only
                  one other pair record causing a vertical RA.>

END vertical_resolution_advisory_not_in_pair_record;
```

```
PROCESS vertical_resolution_advisory_selection;

    <This process examines all pair records in which the subject AC is
     contained.  The effective vertical resolution advisory is selected
     from all of the pair records and placed in the conflict table entry.>

    SET vertical resolution advisory in conflict table to null;
    SET conflict table entry vertical pair record pointer to null;
    SET vertical resolution advisory multiplicity count to zero;

    LOOP:
        Get next pair record associated with this conflict table;
    EXITIF (no more pair records);
        IF (subject AC is in this pair record)
            THEN IF (vertical resolution advisory in this pair record for
                           this AC is not null)
                      THEN save effective resolution advisory in conflict
                                table entry;
                           Increment vertical resolution advisory multiplicity
                                count;
                           IF (conflict table entry pair record pointer EQ null)
                                THEN SET conflict table entry pair record pointer
                                          to point to this pair record;
                                ELSE;
                      ELSE;
            ELSE;
    ENDLOOP;

END vertical_resolution_advisory_selection;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_resolution_advisory_selection;

     CTENTRY.VMAN = $NULLRES;
     CTENTRY.ACIDV = $NULL;
     CTENTRY.MULTV = 0;

     LOOP;
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records);
          IF ((ACID.CTENTRY EQ TPREC.ac1.PAC) OR (ACID.CTENTRY EQ TPREC.ac2.PAC))
               THEN IF (TPREC.ac.PVMAN NE $NULLRES)
                         THEN CTENTRY.VMAN = EFFVRA(TPREC.ac.PVMAN,CTENTRY.VMAN);
                              CTENTRY.MULTV = CTENTRY.MULTV + 1;
                              IF (CTENTRY.ACIDV EQ $NULL)
                                   THEN CTENTRY.ACIDV = TPREC;
                                   ELSE;
                         ELSE;
               ELSE;
     ENDLOOP;

END vertical_resolution_advisory_selection;
```

13. RESOLUTION ADVISORIES EVALUATION ROUTINE

The Resolution Advisories Evaluation Routine (RAER) is called to determine resolution advisories for a pair of aircraft requiring resolution by the Master Resolution Task or to compute resolution advisories for the controller alert function of the Conflict Resolution Data Task. RAER receives as input an Encounter List entry, the altitude separation threshold, a flag indicating whether single or double dimension resolution advisories are requested, and a flag indicating whether the Master Resolution Task (Section 12) or the Conflict Resolution Data Task (Section 11) is calling this routine. A Pair Record is also provided to RAER when it is called by the Master Resolution Task. The routine generates positive or negative horizontal or vertical resolution advisories, positive double dimension resolution advisories or vertical speed limit (VSL) resolution advisories for each aircraft that is to be maneuvered. The major functions of the Resolution Advisories Evaluation Routine are presented in Table 13-1.

RAER provides resolution for a conflict pair by selecting the "best" set of resolution advisories from a predetermined master list of advisories. This master list of advisories is conceptually divided into three groups: resolution advisory sets that maneuver only the first aircraft; sets that maneuver only the second aircraft; and sets that maneuver both aircraft. Tables 13-2 and 13-3 show how the maneuvering aircraft are determined.

Each of these three groups is further divided into three subgroups: resolution advisory sets with horizontal-only advisories; sets with vertical-only advisories; and sets with double dimension advisories.

"Best" is defined as being that set of advisories that meets certain minimum criteria and surpasses the minimum criteria in more ways than any other potential resolution advisory set.

Only positive resolution advisories are included in the master list of resolution advisory sets. Negative advisories are a special case of positives and are signified by setting a flag associated with a resolution advisory set. The negative resolution advisory that replaces a positive resolution advisory is actually the negative of the opposite sense advisory. Thus the negative of climb is don't descend. The negative of turn right is don't turn left.

AD-A104 148 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-2 FAA-RD-81-45-2 NL

2 OF 7
AD
A104148

TABLE 13-1

MAJOR FUNCTIONS PERFORMED BY THE
RESOLUTION ADVISORIES EVALUATION ROUTINE

Resolution Advisories Evaluation Routine

1. Determine which aircraft to maneuver

   - Select direction of vertical resolution advisories if both aircraft are maneuvered

2. Select list of potential resolution advisories that maneuvers the appropriate aircraft

3. Determine if a positive vertical advisory should be modified because of proximity to terrain

4. Calculate predicted separation based on response to potential resolution advisories

5. Determine if the negative sense of any of the resolution advisories is acceptable

   - Modify those resolution advisories for which negatives are acceptable

   - Calculate vertical speed limit advisories as possible replacements for any negative vertical advisories

     - Modify those negative vertical advisories for which vertical speed limits are acceptable

TABLE 13-1
(Concluded)

6. Evaluate absolute features for all potential resolution advisory sets

   - If none of the potential resolution advisory sets has all absolute features set to true, perform multi-aircraft conflict resolution logic

     - If none of the potential resolution advisory sets has all absolute features set to true, indicate no selection of advisories for now

   - If more than one set of advisories has all absolute features set to true, perform relative features evaluation

   - If more than one set of advisories is tied for the "best," perform tie-breaking features evaluation

TABLE 13-2

WHICH AIRCRAFT TO MANEUVER WHEN NEITHER IS IN FINAL APPROACH ZONE

AIRCRAFT 2	AIRCRAFT 1			
	Controlled Equipped	Controlled Unequipped	Uncontrolled Equipped	Uncontrolled Unequipped
Controlled Equipped	Both	AC2	AC1[1]	AC2
Controlled Unequipped	AC1	Neither	AC1	Neither
Uncontrolled Equipped	AC2[1]	AC2	Both	AC2
Uncontrolled Unequipped	AC1	Neither	AC1	Neither

[1]Both aircraft will be maneuvered if PIFR is set.

TABLE 13-3

WHICH AIRCRAFT TO MANEUVER WHEN AIRCRAFT 2 IS IN FINAL APPROACH ZONE

AIRCRAFT 2	AIRCRAFT 1			
	Controlled Equipped	Controlled Unequipped	Uncontrolled Equipped	Uncontrolled Unequipped
Controlled Equipped	AC1	AC2	AC1	AC2
Controlled Unequipped	AC1	Neither	AC1	Neither
Uncontrolled Equipped	AC1	AC2	AC1	AC2
Uncontrolled Unequipped	AC1	Neither	AC1	Neither

Rule to determine which aircraft to maneuver:

If one of the aircraft is on final approach:

1. Give resolution advisories to the aircraft not on final approach if it is equipped.

2. Give resolution advisories to the aircraft on final approach if the other aircraft is unequipped.

Besides including only positive resolution advisories in the master list of advisory sets, another restriction is placed on the vertical resolution advisories for the sets with both aircraft maneuvered. Rather than including all four possible vertical-only advisory sets, and therefore 16 sets of double dimension advisories, only one vertical-only advisory set is included. This is done by allowing the vertical resolution advisories to be selected dynamically for the case in which both aircraft are maneuvered. The master list of resolution advisories and its associated data structure are discussed in Section 13.1. By keeping the master list of advisories as small as possible, the computation time for RAER is minimized.

13.1 Resolution Advisory Data Structure (RADS)

The data structure for a resolution advisory set is described in pseudocode in Section 13.5. Some of the data fields describe intrinsic properties of each resolution advisory set and are "hardwired," while others depend on the encounter and are computed by RAER. The fields that are "hardwired" are shown in Table 13-4.

When both aircraft are expected to respond to resolution advisories, there is normally one "best vertical" resolution advisory set. This "best vertical" resolution advisory set can be found by projecting the aircraft ahead eight seconds and giving the aircraft on top a climb and the one below a descend. For the resolution advisories that maneuver both aircraft in the vertical dimension, the vertical maneuvers are not "hardwired," but are computed using this "eight second rule." The same is not done when only one aircraft is maneuvered because it may be desirable to maneuver one aircraft vertically toward another to avoid a vertical chase.

The "eight second rule" as described above is used only on the first scan in which vertical resolution advisories are selected for a pair of maneuvering aircraft. Once vertical advisories have been selected, on subsequent scans the same vertical sense must be maintained.

The selection of the "best vertical" resolution advisories for a pair of aircraft should not be confused with the selection of the "best" resolution advisory set for a conflict pair. The selection of the "best vertical" advisories only determines which vertical advisory should be considered for each aircraft to resolve the conflict.

TABLE 13-4

FIELDS IN THE RESOLUTION ADVISORY DATA
STRUCTURES THAT ARE "HARDWIRED"

RESOLUTION ADVISORY DATA STRUCTURE FIELD	INITIAL VALUE
CMDED_CMDED	Set if this advisory set contains advisories for both AC; reset otherwise
CMDED_UNCMDED	Set if this advisory set contains advisories for the first AC only, reset otherwise
HORIZ	Set if this advisory set contains a horizontal advisory; reset otherwise
H1	Horizontal resolution advisory for first AC, or null if this AC not maneuvered
H2	Horizontal resolution advisory for second AC, or null if this AC not maneuvered
INDEX1	Set to reference the appropriate entries in the PSEP, QSEP, and HMD matrices that correspond to the horizontal advisory in this advisory set for the first AC
INDEX2	Set to reference the appropriate entries in the PSEP, QSEP, and HMD matrices that correspond to the horizontal advisory in this advisory set for the second AC
INDEX3	Set to reference the appropriate entries in the PSEP, QSEP, VMDA, and VMDB matrices that correspond to the vertical advisory in this advisory set for the maneuvered AC, regardless of whether one or both AC are maneuvered
MATPTR	Set to point to the set of separation matrices to be used with the resolution advisory set. In Two-aircraft Resolution logic there is only one set of matrices. There may be two sets of matrices in the Multi-aircraft Resolution logic.

TABLE 13-4
(Concluded)

RESOLUTION ADVISORY DATA SET FIELD	INITIAL VALUE
NXTADV	Points to next data structure in list
SINGLE	Set if this advisory set has only single dimension advisories; reset if this advisory set has double dimension advisories
UNCMDED_CMDED	Set if this advisory set contains advisories for the second AC only; reset otherwise
VERT	Set if this advisory set contains a vertical advisory; reset otherwise
V1	Vertical resolution advisory for first AC if only first AC maneuvered, null if first AC not maneuvered, or uninitialized if both AC maneuvered
V2	Vertical resolution advisory for second AC if only second AC maneuvered, null if second AC not maneuvered, or uninitialized if both AC maneuvered

13.2 Predicted Separation Calculations

The predicted separation matrices contain the separations that two conflicting aircraft are expected to achieve by responding to resolution advisories. The separation values are computed by performing a fast-time simulation and modeling the performance of the aircraft. This simulation begins with a short delay period (to account for communication and pilot response delays), during which only sensed turns and previous advisories are modeled. This is followed by a maneuver period, during which the aircraft are modeled as responding to the various resolution advisory sets under consideration for the present conflict.

During the fast-time simulation, the minimum values for three-dimensional (3-D) separation (slant range), two-dimensional horizontal separation (range), and vertical separation are recorded in the predicted separation matrices for the various combinations of maneuvers modeled for the aircraft. The 3-D closest approach point, horizontal closest approach point, and vertical closest approach point may occur at different times.

13.2.1 Predicted Separation Data Structures

The predicted separation data structures consist of five matrices: HMD, VMDA, VMDB, PSEP, and QSEP. The first four of these matrices contain minimum separation values: horizontal separation in HMD, vertical separation in VMDA and VMDB, and 3-D separation in PSEP. The QSEP matrix contains 3-D separation values for a particular instant in time.

The HMD matrix is a 3x3 array. Each element represents the minimum horizontal separation for a particular combination of horizontal flight paths for the two aircraft. For computational efficiency, the square of the range is stored, in units of nmi^2. The first dimension of HMD corresponds to the three horizontal maneuvers turn left (TL), continue straight (CS), and turn right (TR) for one aircraft. The second dimension corresponds to the same three horizontal maneuvers for the other aircraft. Each of the three horizontal maneuvers for one aircraft combines with those for the other aircraft, giving a total of nine combinations. These nine combinations model all possible horizontal resolution advisory sets. Negative horizontal resolution advisories are not explicitly modeled, but are considered to be represented by the CS path.

All nine horizontal combinations are not always formed. When only one aircraft is to be maneuvered, only the CS path is modeled for the other aircraft. If an aircraft has a positive horizontal advisory in the HMAN field of its Conflict Table Entry, then the path corresponding to the opposite-sense positive advisory need not be modeled, since such an incompatible advisory cannot be selected. For instance, if an aircraft has a previous turn left advisory in its HMAN field, then the turn right path need not be modeled. In the case of previous negative horizontal advisories, however, no paths can be eliminated, since they are all needed by the negative resolution advisory determination logic.

The VMDA matrix is a one-dimensional three-element array. Each element represents the minimum vertical separation for a particular combination of vertical flight paths for the two aircraft. Unlike the horizontal advisories, not all combinations of vertical resolution advisories are valid; therefore, only a total of three combinations is considered. (Each of these three combinations will hereafter be referred to as a vertical "level.")

Vertical level one always represents both aircraft projected ahead with their current vertical rates. The meaning of the other two levels will depend upon which aircraft are to be maneuvered. If both aircraft are to be maneuvered, then level two will correspond to the vertical resolution advisories picked by the "eight second rule," as described previously in Section 13.1, and level three will correspond to the negative of these resolution advisories. Note that negative vertical advisories are explicitly modeled in this case. If only one of the aircraft is to be maneuvered, levels two and three take on different meanings. In this case, vertical level two will represent a descend advisory for the maneuvered aircraft, and level three will represent a climb advisory for that aircraft. The unmaneuvered aircraft will be projected ahead with its current vertical rate for all three levels. Negative vertical advisories are not explicitly modeled in this case.

There is one exception to the above rules for determining vertical levels. When it is desired to model a descend for an aircraft which is less than a distance ATERN above the terrain threshold, a don't climb is modeled instead. Table 13-5 summarizes the definitions of the three vertical levels for all of the cases described above.

TABLE 13-5

PSEP VERTICAL LEVELS

Both Aircraft Maneuvered

LEVEL 1:	Project both aircraft ahead with their current vertical rates.
LEVEL 2:	Project each aircraft ahead following positive vertical resolution advisory chosen by "eight second rule."[1]
LEVEL 3:	Project each aircraft ahead following negative of vertical resolution advisory chosen by "eight second rule."

Only One Aircraft Maneuvered

LEVEL 1:	Project both aircraft ahead with their current vertical rates.
LEVEL 2:	Descend[1] for maneuvered aircraft, project unmaneuvered aircraft ahead at current vertical rate.
LEVEL 3:	Climb for maneuvered aircraft, project unmaneuvered aircraft ahead at current vertical rate.

[1]Descend is modeled as don't climb if aircraft is less than ATERN above the terrain threshold.

The PSEP matrix is a 3x3x3 array. Each element represents the minimum 3-D separation for a particular combination of both horizontal and vertical flight paths for the two aircraft. In computing the separation values, the vertical separation component is weighted by a factor of VWEIGHT. As was the case with the HMD matrix, the square of each separation value is stored, in units of nmi^2. The first two dimensions of PSEP are the same as the dimensions of HMD. That is, they represent the horizontal flight paths which are modeled for the two aircraft. The third dimension corresponds to the three vertical levels, as previously described for the VMDA matrix. For modeling vertical-only resolution advisories, aircraft are considered to follow the CS path horizontally.

The VMDB matrix is also a one-dimensional three-element array. Like VMDA, each element contains a vertical separation value for one of the three vertical levels. Unlike VMDA, however, the elements of VMDB represent achievable separations, rather than absolute minimum separations, for single vertical advisories. Each value is computed to be the unweighted vertical component of the minimum 3-D separation for the vertical-only maneuvers represented by the level.

The QSEP ("quick separation") matrix is another 3x3x3 array. Each element is a vertical-weighted 3-D separation value, and the dimensions of QSEP are defined the same as for the PSEP matrix. When the aircraft have been modeled as responding to new resolution advisories for a short period of time (defined by the QTIME parameter), the instantaneous 3-D separation values are saved in the QSEP matrix. Thus, the QSEP matrix represents a "snapshot" of separation values shortly after the aircraft have begun to maneuver in response to resolution advisories. QSEP values are occasionally used as a final tie-breaker in choosing the best resolution advisory set.

13.2.2 Modeling of the Delay Period

In order to account for communication and pilot response delays, the modeling of the aircraft begins with a short delay period. The delay period models the aircraft for a constant length of time, specified by the DELAY parameter. During this period it is assumed that the advisories being considered for the present conflict will not yet be effective. Therefore, only turns strongly sensed by the tracker and resolution advisories issued previously will affect the modeled flight paths. If any such sensed turn or previous advisory is in effect for either aircraft, then nonlinear flight is assumed. In this case, the delay period is modeled with a fast-time simulation; the length of each time step is specified by the DELINT parameter. (DELAY should always be an even multiple of DELINT.) During the first half of the delay period, any sensed turns are modeled for each aircraft. During the last half of the delay period, the aircraft are modeled as responding to any resolution advisories which are currently being displayed (as recorded in the VMAND and HMAND fields of the Conflict Table Entries). Such previous advisories are modeled only during the delay period.

The modeling of sensed turns and previous turn left and turn right advisories during the delay period is performed by assuming a constant bank angle, defined by the parameter BANKA. Previous negative horizontal advisories are not explicitly modeled and do not contribute to nonlinear flight. Previous vertical resolution advisories are modeled by determining a "final" vertical rate which the aircraft is to achieve. If this final vertical rate is different from the aircraft's current rate, the aircraft is accelerated toward the final rate using one of the parameters ACCELC or ACCELD. In the case of positive vertical advisories, the final rate is determined by one of the parameters ZDUPF, ZDUPS, ZDDWNF, or ZDDWNS, depending on the directional sense of the advisory and the aircraft's current velocity. In the case of negative vertical resolution advisories (including VSLs), the previous advisory may contain both upward and downward components. Here the current vertical rate and a pair of vertical rate limits (chosen from Table 13-6) determine the final vertical rate.

In the event that neither aircraft has a sensed turn nor a previous resolution advisory, then linear flight is assumed during the delay period. In this case, both aircraft are simply projected straight ahead for DELAY seconds, using their current velocity components.

During the modeling of the delay period, only one flight path is modeled for each aircraft. The result of this modeling is a set of four minimum separation values: PSEPI, HMDI, VMDAI, and VMDBI. These values are used to initialize all elements of the PSEP, HMD, VMDA, and VMDB arrays, respectively, for the modeling of the maneuver period.

13.2.3 Modeling of the Maneuver Period

The maneuver period is modeled with a fast-time simulation, in a series of time steps. The length of each step is given by the TIMINT system parameter, which is chosen to be an integral divisor of the sensor scan time (SCANT). During the maneuver

TABLE 13-6

MAXIMUM AND MINIMUM VERTICAL RATE PARAMETERS FOR MODELING NEGATIVE VERTICAL RESOLUTION ADVISORIES

DON'T CLIMB OR LIMIT CLIMB RESOLUTION ADVISORY	MAXIMUM VERTICAL RATE (ZDMAX)
Don't Climb	0
Limit Climb 500 ft/min	V500
Limit Climb 1000 ft/min	V1000
Limit Climb 2000 ft/min	V2000

DON'T DESCEND OR LIMIT DESCENT RESOLUTION ADVISORY	MINIMUM VERTICAL RATE (ZDMIN)
Don't Descend	0
Limit Descent 500 ft/min	-V500
Limit Descent 1000 ft/min	-V1000
Limit Descent 2000 ft/min	-V2000

period, multiple flight paths are modeled for each aircraft, as previously explained. The expected response of the aircraft to each resolution advisory is modeled, and the minimum separation values between the aircraft are computed. Resolution advisories are modeled in the same way as described for previous advisories during the delay period (see Section 13.2.2), except that VSLs are never modeled during the maneuver period. As the predicted positions and velocities are determined, certain of these values are saved in the Resolution Advisory Projected Position (RAPP) Table to be used by the Domino Feature (see Section 13.4.2.4). All elements of the RAPP Table are set to an uninitialized value before the predicted separation calculations begin. This is done so that if the domino logic tries to use values that have not been calculated, this condition can be recognized and the projection calculations may then be performed.

When QTIME seconds have passed in the modeling of the maneuver period, the instantaneous values of 3-D separation for all combinations of flight paths are saved in the QSEP matrix, as previously explained. These values are used in certain cases as a final tie-breaker in determining the best resolution advisories (see Section 13.4.3).

The maneuver period is modeled for a fixed length of time, MANTM. MANTM is calculated basically as follows: First, the time to 3-D closest approach (vertical weighted) is calculated after the delay period, assuming straight flight for each aircraft. To this value a parameter (TCADEL) is added. This result is replaced by a fixed value if the aircraft are slow-closing. Next, an upper limit is applied. This limit is computed so as to not allow either aircraft to be modeled through a turn of more than TURNA1, nor to allow both aircraft to be modeled through a combined turn of more than TURNA2. Finally, fixed upper and lower limits of MTUL and MTLL, respectively, are applied.

For some geometries the aircraft will still be coverging at the end of MANTM. For these geometries, the measured minimum separation will be larger than the true closest approach. Separate tests are applied to determine 3-D, horizontal, and vertical convergence.

The VMDA matrix is used for determining if negative resolution advisories give sufficient separation. Negative vertical resolution advisories should not be issued that will allow the aircraft to converge. Therefore, an entry in VMDA is set to zero if vertical convergence is indicated. Likewise, if three-dimensional convergence is indicated at MANTM, then PSEP is set to zero if the resolution advisory set contains a horizontal maneuver. For vertical maneuvers only, PSEP is calculated by a three-dimensional miss distance formula. Similarly, if horizontal convergence is indicated at MANTM, then HMD is computed from a horizontal miss distance formula for the center element of the HMD matrix (no horizontal maneuvers) or set to zero for any other element (at least one horizontal maneuver).

13.3 Negative Resolution Advisories Evaluation

The RAER logic selects positive resolution advisories in the horizontal or vertical dimension and then, in some cases, modifies those resolution advisories to negative. If RAER determines that negative resolution advisories provide sufficient separation, the NEGATIVE flag is set in the Resolution Advisory Data Structure.

13.3.1 Horizontal and Vertical Negative Resolution Advisories

Reference to the negative of a resolution advisory always means the negative of the opposite direction advisory. That is, the negative of a turn left is don't turn right. The negative of turn right is don't turn left. The negative of climb is don't descend, and the negative of descend is don't climb.

When the vertical dimension has been selected for resolution, negative vertical resolution advisories are selected if the vertical predicted separation at the time a pilot responds is greater than the positive resolution advisory altitude separation (ASEP) and the aircraft will not converge to less than ASEP during the projection interval. If both aircraft are maneuvered, negative vertical advisories are explicitly modeled. This is not true if only one aircraft is maneuvered. If only one aircraft is maneuvered, both the climb and descend advisories will be examined. If the advisory will maneuver that aircraft into the unmaneuvered aircraft, then the negative of that advisory is not acceptable, since the negative advisory would still allow the aircraft to maneuver into the unmaneuvered aircraft. If the advisory will maneuver the aircraft away from the unmaneuvered aircraft, then the separation achieved by the positive advisory is checked. If the positive sense of the advisory will prevent the aircraft from coming closer than ASEP, then the negative is acceptable, since the negative is essentially a level-off advisory in this particular altitude geometry. The negative is acceptable only if the unmaneuvered aircraft does not have a large vertical rate towards the maneuvered aircraft.

When the horizontal dimension has been selected for resolution, negative horizontal resolution advisories are selected if the horizontal predicted separation along each of the possible response paths is greater than the positive resolution advisory horizontal miss distance threshold (MDTHSQ). To check if negative horizontal resolution advisories give sufficient separation, four HMD values must be examined if both aircraft are maneuvered, and two HMD values must be examined if only one aircraft is maneuvered.

If both aircraft are maneuvered, negative advisories are not selected unless the predicted separations are greater than MDTHSQ for all of the following cases:

1. Both aircraft perform the positive maneuver being evaluated

2. Either aircraft performs the positive maneuver; the other continues straight

3. Neither aircraft performs the positive maneuver

For example, if turn left/turn left is the advisory set being examined, then the HMD values of turn left/turn left, turn left/continue straight, continue straight/turn left and continue straight/continue straight must all be greater than MDTHSQ if the NEGATIVE flag is to be set indicating a don't turn right/don't turn right advisory combination is acceptable for resolution.

If only one aircraft is maneuvered, the predicted separation values of two HMD entries are checked. If turn left is the advisory being examined, then the HMD value of the turn left/continue straight and continue straight/continue straight advisories are checked to determine if a don't turn right would be a sufficient advisory to the maneuvered aircraft. The value of MDTHSQ is modified (increased) if either aircraft is turning.

Double dimension advisories are not checked for the possibility of giving negative advisories. When single dimension advisories are checked for negatives being sufficient, a check for positive or negative advisories to both aircraft (assuming both maneuvered) is always made. The logic will not issue a positive advisory to one aircraft and a negative to the other, except for the one case where the aircraft to receive a descend advisory is less than ATERN feet above the terrain threshold. In this case the BELOW1000 flag is set, indicating that the descend should be changed to a don't climb. The H1, V1, H2, and V2 fields of the selected resolution advisory should be modified, if necessary, because of the NEGATIVE flag or the BELOW1000 flag.

13.3.2 Vertical Speed Limit (VSL) Advisories Evaluation

If negative vertical resolution advisories are selected for both aircraft, additional logic determines if a limit vertical rate advisory can be used to resolve the conflict. That is, will a "limit descent to some rate" suffice versus a don't descend (i.e. limit vertical rate to zero). VSL advisories are considered to be a subset of negative vertical resolution advisories.

The desired vertical speed limit is computed based on the current altitude separation, current vertical velocity, expected pilot delay time, and desired separation at the projected closest separation time. Speed limits are computed for each aircraft assuming that there is no change in the direction and velocity of the other aircraft. To receive a VSL, an aircraft must be maneuvering vertically faster than the minimum rate (MRATE) and the direction of the aircraft's current vertical velocity must be towards the other aircraft.

VSLs are computed individually for each aircraft of the pair. Consequently, only one or both may receive VSLs or different VSLs may be given to each one. The computed VSL is rounded down to 2000 ft/min, 1000 ft/min, or 500 ft/min. If a VSL resolution advisory is selected, it is assigned to the vertical field in the Resolution Advisory Data Structure.

13.4 Features Evaluation

After determining which subset of the master list of potential resolution advisories is applicable to the subject conflict pair, the subset must then be reduced to a single advisory set that will resolve the conflict. The first step in this reduction process was calculating the projected separations in response to the various resolution advisories. The projected separations were then used to determine if the negative of any of the single dimension advisory sets will provide sufficient separation. Then the projected separations and the response paths are used to determine which advisory set will resolve the conflict better than the other advisory sets.

The resolution advisory sets are evaluated by applying a number of sequential tests, called features. If a feature is true (i.e. passes the test) for a given advisory set then the feature is said to be "set." The outcome of the tests may depend on the geometry of the encounter, the speeds of the aircraft, the predicted separation, or many other factors. The pseudocode for the Relative Features Evaluation Process in Section 13.5 shows these tests in order of precedence. Tables 13-7 and 13-8 provide the logic for the resolution advisory compatibility and reinforcement checks, which are used by some of the features.

Although alternative implementations are possible, this discussion refers to the tests as individual routines that operate on the list of resolution advisories. Each test has a weight associated with it, the most important test having the highest weight. These weights are stratified so that the weight of a test is greater than the sum of the weights for the less important tests. This could be accomplished by using sequential powers of two for the weights. When a resolution advisory satisfies or passes a test or feature, its VALUE field is increased by the weight for that test. The resolution advisory with the highest number in its VALUE field is considered the best resolution advisory. The RADS data structure is general enough to allow an efficient implementation in most programming languages. The implementation should be made flexible enough to allow new tests to be added and the list reordered without a major redesign.

To reduce computation time, the list could be pruned after some of the tests. For example, the test that decides whether to favor single or double resolution advisories is guaranteed to cut the list to about half of its size, if all of the previous tests are equal. By eliminating all of the resolution advisories that are not tied with the highest value, the amount of computer processing time could be reduced. Whether or not this savings is significant depends on the implementation. This pruning of the list can be done only after all of the features with higher weights have been evaluated.

13.4.1 Absolute Features

The first three features, the Deliverable, Dimension Available and Manuevered/Unmanuevered Conflict Features, are called absolute features. As the name absolute implies, each of these

TABLE 13-7

RESOLUTION ADVISORY COMPATIBILITY LOGIC

NEW RESOLUTION ADVISORY	PREVIOUS RESOLUTION ADVISORY										
	TL	TR	DTL	DTR	DTL& DTR	CL	DCL/ LIMCL	DES	DDES/ LIMDES	DCL& DDES	NO RES ADV
TL	1[1]	0	0	1	0	1	1	1	1	1	1
TR	0	1	1	0	0	1	1	1	1	1	1
DTL	0	1	1	1	1	1	1	1	1	1	1
DTR	1	0	1	1	1	1	1	1	1	1	1
CL	1	1	1	1	1	0	0	0	1	0	1
DCL/LIMCL	1	1	1	1	1	0	1	1	1	1	1
DES	1	1	1	1	1	1	1	1	0	0	1
DDES/LIMDES	1	1	1	1	1	1	1	0	1	1	1
NO RES ADV	1	1	1	1	1	1	1	1	1	1	1

[1]1 = Compatible, 0 = Incompatible
Each dimension of double dimension resolution advisory sets is tested separately.
Both dimensions must be compatible for the set to be compatible.

TABLE 13-8

RESOLUTION ADVISORY REINFORCEMENT LOGIC

NEW RESOLUTION ADVISORY	PREVIOUS RESOLUTION ADVISORY										
	TL	TR	DTL	DTR	DTL& DTR	CL	DCL/ LIMCL	DES	DDES/ LIMDES	DCL& DDES	NO RES ADV
TL	1[1]	0	0	1	0	0	0	0	0	0	0
TR	0	1	1	0	0	0	0	0	0	0	0
DTL	0	1	1	0	1	0	0	0	0	0	0
DTR	1	0	0	1	1	0	0	0	0	0	0
CL	0	0	0	0	0	1	0	0	1	0	0
DCL/LIMCL	0	0	0	0	0	0	1	1	0	1	0
DES	0	0	0	0	0	0	1	1	0	0	0
DDES/LIMDES	0	0	0	0	0	1	0	0	1	1	0
NO RES ADV	0	0	0	0	0	0	0	0	0	0	0

[1] 1 = Reinforces, 0 = Does not reinforce
Each dimension of double dimension resolution advisory sets is tested separately.
If either dimension reinforces the previous set, then the new advisory set reinforces the previous advisory set.

features must be set true for the resolution advisory set that is selected to resolve the conflict pair. If none of the advisory sets has all three absolute features set, then no advisory set is selected to resolve the conflict.

13.4.1.1 Resolution Advisory Compatibility and Deliverability

The Deliverable Feature determines if the resolution advisories can be delivered to the aircraft. This feature is set true if, on the first scan that advisories are selected for a conflict pair, either the advisories are flagged as negatives or they provide a minimum separation greater than that which would be obtained if no advisories were sent to the aircraft. On subsequent scans of advisory selection, this feature is automatically set true.

The Dimension Available Feature determines if the resolution advisories will be accepted by the ATARS avionics. This feature is set true only if the advisories for both aircraft are compatible with all other advisories previously selected for each of the aircraft. The compatibility logic in Table 13-7 is used by this feature.

The Manuevered/Unmanuevered Conflict Feature checks for a maneuvered aircraft in the current conflict being unmanuevered in another conflict pair. If this situation is detected, then this feature is reset for any advisory sets that have a component in the same dimension as that used to resolve the previous conflict. This is done because the resolution for the previous pair was based on one of the aircraft not maneuvering. It is not yet known how a maneuver would affect the previous conflict.

The Deliverable Feature is the only absolute feature evaluated when RAER is called by the Conflict Resolution Data Task. When RAER is called from the Master Resolution Task, all three absolute features must be evaluated.

If either or both of the aircraft in a conflict are also in one or more other conflicts, then it is possible that resolution advisories given to the aircraft for the other conflicts restrict the selection of advisories for the current conflict to the point where none of the potential resolution advisory sets has all of the absolute features set. If this occurs, then the multi-aircraft resolution logic is used to attempt to select resolution advisories for the pair. The multi-aircraft resolution logic will not be performed when RAER is called by the Conflict Resolution Data Task. This is ensured by overriding the outcome of the Deliverable Feature, if this is necessary to obtain a resolution advisory set with all absolute features set to true. The multi-aircraft logic can be performed only if both aircraft are to be maneuvered. Otherwise all possible resolution advisory sets have already been examined.

13.4.1.2 Multi-aircraft Resolution Logic

The multi-aircraft resolution logic first determines why no advisory sets have all absolute features set. For any advisory sets with the Deliverable and Dimension Available Features set, the Maneuvered/Unmaneuvered Conflict Feature is evaluated using the multi-aircraft definition. If all absolute features are set true for any advisory sets after the re-evaluation, then resolution advisories can be given this scan. Otherwise, additional logic is executed in an attempt to increase the number of advisory sets being examined for selection.

If only one aircraft is to be maneuvered for the current conflict, then all possible resolution advisory sets have been examined. Attempts to resolve the pair are delayed until later this scan or until the next scan. However, RAER will not be able to select advisories for the current conflict pair until the conditions preventing advisory selection change.

If both of the aircraft in the current conflict are to be maneuvered, then the multi-aircraft resolution logic looks at vertical advisories other than the "best vertical" set. If the aircraft are currently within ZCARE feet vertically, then vertical advisories opposite to the "best vertical" are examined. If the aircraft are currently separated by more than ZCARE, both aircraft are given the same vertical advisory. Both climb and descend to each aircraft are examined. All possible double dimension advisory sets are also considered.

The absolute features for the new advisory sets are then evaluated using the multi-aircraft definition of the Maneuvered/Unmaneuvered Conflict Feature. If any advisory set has all absolute features set, then resolution advisories will be selected this scan. Otherwise, resolution is delayed.

13.4.2 Relative Features

The following features are called relative features. They are used only to select one potential resolution advisory set over another as the "better" set of resolution advisories. None of these features must necessarily be set true for the advisory set that is selected to resolve the pair.

The relative features evaluate such things as the projected separation against certain thresholds, the current separations against certain thresholds, and the current horizontal or vertical velocities against the other aircraft or against certain thresholds. Also evaluated is an advisory's ability to reinforce the advisory given to this pair on the previous scan, advisories given to either aircraft from other ATARS sites or BCAS, and the turn status of the aircraft.

13.4.2.1 Predicted Separation Dependent Features

Some of the features examine the predicted separation in response to advisories. The top priority relative feature (PSEP GE SEP1) checks for separation greater than a minimum desirable separation. Any advisory set providing at least this separation has this feature set.

A lower priority feature (PSEP GE SEP2) checks for a larger predicted separation, SEP2. There are two SEP2 values, which are computed dynamically. One value is used for single dimension advisories and the other for double dimension advisories. Each value is a percentage of the maximum separation provided by any single (double) dimension advisory with all absolute features set.

Two low priority features consider the predicted miss distance in the resolution dimension. If a vertical advisory set will provide large separation, the Big Vertical Miss Distance Feature is set. If a horizontal advisory set will provide large separation, the Big Horizontal Miss Distance feature is set. These two features provide the only exception to the rule that all features have different weight. If either feature is set, the other feature is also set. This effectively gives both features the same weight.

13.4.2.2 Aircraft Geometry and Velocity Dependent Features

Certain relative features consider the conflict pair's geometry. If one aircraft is not maneuvered and that aircraft has a large vertical velocity or a speed much greater than that of the other aircraft, then the Unmaneuvered With Large Vertical Rate Feature or the Fast Unmaneuvered/Slow Maneuvered Feature is set for those resolution advisory sets containing double dimension advisories.

Another feature tests the speed of each aircraft. If either maneuvered aircraft has a large velocity or if both aircraft are slow, then all PRAs with horizontal advisories have the Speed Check Feature set.

If either of the aircraft is receiving a terrain or obstacle alert, the Terrain Or Obstacle Alert Feature is set true for all horizontal-only resolution advisory sets.

13.4.2.3 Aircraft Maneuverability Dependent Features

Some of the features consider the aircraft's maneuverability with respect to horizontal turn status and vertical rate status and with respect to previous resolution advisories.

If either of the maneuvered aircraft has a previous horizontal advisory, then the Reinforce Resolution Advisory From Non-Connected Site or BCAS Feature or the Reinforces Prior Resolution Advisories Feature is set true for any advisory set with a compatible horizontal resolution advisory. Similarly, the Reinforces Turn Feature is set true for any advisory set with a horizontal maneuver that reinforces a sensed turn for either maneuvered aircraft. The logic in Tables 13-9 and 13-10 is used by these and other features to determine compatibility of horizontal advisories with turn status and vertical advisories with vertical velocity.

13.4.2.4 Domino Feature

When an aircraft is given a resolution advisory, it is possible that by executing that maneuver, the aircraft will be directed into another conflict requiring resolution advisories. This type of conflict, caused by a resolution advisory, is called a domino conflict. If the second conflict begins before the first conflict is resolved, then there is a multi-aircraft conflict. It is always desirable to avoid domino-created multi-aircraft conflicts, if at all possible. A way to avoid a domino-caused multi-aircraft conflict is to model an aircraft's response to a resolution maneuver and determine if a conflict requiring resolution advisories will be created with another aircraft during the time the aircraft is responding to the resolution advisory. Then, if there is more than one set of acceptable resolution advisories for resolving a conflict, the set of resolution advisories that does not cause a domino multi-aircraft conflict should be the set of resolution advisories chosen. Logic that performs the checks for detecting a domino-caused multi-aircraft conflict is called the domino logic.

TABLE 13-9

AIRCRAFT TURN STATUS VERSUS
HORIZONTAL RESOLUTION ADVISORY COMPATIBILITY LOGIC

AC TURN STATUS	HORIZONTAL RESOLUTION ADVISORY					
	$TL	$TR	$DTR	$DTL	$NULLRES	$NORES
$STRNGLFT	$TRUE[1]	$FALSE	$TRUE	$FALSE	$TRUE	$TRUE
$STRNGRGT	$FALSE	$TRUE	$FALSE	$TRUE	$TRUE	$TRUE
ALL OTHERS	$TRUE	$TRUE	$TRUE	$TRUE	$TRUE	$TRUE

[1]$TRUE - compatible
$FALSE - incompatible

TABLE 13-10

AIRCRAFT VERTICAL VELOCITY VERSUS
VERTICAL RESOLUTION ADVISORY COMPATIBILITY LOGIC

AC VERTICAL	VERTICAL RESOLUTION ADVISORY					
VELOCITY	$CL	$DES	$DDES	$DCL	$NULLRES	$NORES
GREATER THAN ZDTH	$TRUE[1]	$FALSE	$TRUE	$FALSE	$TRUE	$TRUE
BETWEEN AND INCLUDING -ZDTH & ZDTH	$TRUE	$TRUE	$TRUE	$TRUE	$TRUE	$TRUE
LESS THAN -ZDTH	$FALSE	$TRUE	$FALSE	$TRUE	$TRUE	$TRUE

[1]$TRUE - compatible
$FALSE - incompatible

The domino logic is called by the RAER during evaluation of the relative features. Two features are controlled by the outcome of the domino logic. The most desirable situation, and therefore the higher priority of the domino features, is for neither aircraft to be predicted as being involved in a domino conflict because of the subject resolution advisories. The next domino feature is set if only one aircraft is predicted to be in a domino conflict because of response to its resolution advisory. The remaining possibilities are that both aircraft are predicted to be in a domino conflict because of their resolution advisories, or that domino logic is not performed for this pair of aircraft. In these two cases, neither of the domino features is set.

The domino logic must determine all potential resolution advisories available to each aircraft. The potential resolution advisories are needed by the Domino Coarse Screen Filter (Section 13.4.2.4.2) to determine the extent of the search limits. This logic selects all aircraft that are within the search limits and creates a Potential Domino Conflict List for each of the aircraft requiring resolution advisories.

To determine if a given resolution advisory will cause an aircraft to come into conflict with another aircraft, the aircraft's path in response to the resolution advisory must be modeled. This was done when the predicted separation calculations were performed. The projected positions and velocities at four points (one, two, three, and four scans after the delay period) were stored in the Resolution Advisory Projected Position (RAPP) Table. The domino logic compares these values to the projected positions and velocities of aircraft from the Potential Domino Conflict List using a shortened detection logic. (Previous resolution advisories and tracker sensed turns will be modeled for aircraft on the Potential Domino Conflict List in the same way that these maneuvers were modeled for the subject aircraft.) Since the only concern of the domino logic is for a conflict requiring resolution advisories being created, the resolution advisory checks are the only checks of the detection logic performed. If a domino conflict is determined with any aircraft on the Potential Domino Conflict List, then the remainder of the list need not be checked for another domino conflict caused by the same resolution advisory. The subject resolution advisory is flagged as causing a domino conflict. The domino checks then begin for the next resolution advisory.

The first check performed in the domino logic is to determine which aircraft is (are) maneuvered. This can be done by examining the CMDED_UNCMDED and the UNCMDED_CMDED flags of the first potential resolution advisory. This information is used in the Domino Coarse Screen Filter.

The Potential Resolution Advisory Domino Status Variables are set by cycling through all the potential resolution advisories that have all absolute features set and whose total value is tied for the highest valued resolution advisory set. There is only one status variable for negative horizontal resolution advisories, since a don't turn left and don't turn right are both equivalent to continue straight.

The Domino Coarse Screen Filter determines a list of potential domino conflict aircraft for each of the subject aircraft that is to receive a resolution advisory. The Domino Detection Filter checks each potential resolution advisory for causing a domino conflict with an aircraft on the Potential Domino Conflict List.

13.4.2.4.1 Domino Logic Data Structures

There are two data structures used by the domino logic. Both are described in the pseudocode in Section 13.5. One data structure (PRADSVVBL) is associated with each maneuvering aircraft in the subject conflict pair. The other data structure (PDC_LIST) is associated with each aircraft on the Potential Domino Conflict (PDC) List.

The data structure associated with the subject aircraft contains information on the potential resolution advisories for that aircraft and their status in terms of the domino logic. All of the Potential Resolution Advisory Status Variables are initialized to $NOTPRA (not a potential resolution advisory). As the Domino Coarse Screen Filter examines the RADS, those status variables corresponding to RADS that have the potential to be selected for resolution are set to $DOMNP (domino not yet processed). Those advisories with status $DOMNP are used to determine the domino coarse screen search limits.

As the Domino Detection Filter processes the aircraft on the PDC List, the status variables having the state $DOMNP transition to the state $DOMCNC (domino conflict not caused by this resolution advisory) or $DOMCC (domino conflict caused by this resolution advisory) as appropriate. Then the domino features are set for each RADS based on the final state of the status variables.

The second data structure is associated with each aircraft on the Potential Domino Conflict List. The data structure points to the State Vector of the aircraft, where the Domino Object Projected Positions (DOPP) values are found. These values are used in the Domino Detection Filter. The data structure also points to the next aircraft on the PDC list.

The data structure for the PDC aircraft contains resolution advisory status varibles. These variables are initialized to $NOTTEST (this advisory not tested for causing a domino conflict with this aircraft). After domino detection is performed between a subject and object aircraft for a resolution advisory, the appropriate status variable is set to either $NODOMC (no domino conflict caused by this advisory with this aircraft) or $DOMC (domino conflict caused by this advisory with this aircraft).

13.4.2.4.2 Domino Coarse Screen Filter

The Domino Coarse Screen Filter determines a list of potential domino conflict aircraft for each maneuvered aircraft of the subject conflict pair. This is done by calculating the maneuvered aircraft's projected response path to all of the potential resolution advisories during the duration of the conflict, and adding to this the maximum immediate separation threshold distance.

The Domino Coarse Screen Filter performs a forward and backward search along the X-list or EX-list. The distance to be searched along the list is a function of the current speed and heading of the subject aircraft and the potential resolution advisories. The lists searched are also a function of the subject aircraft being on the X-list or EX-list. If the subject aircraft is on the X-list, only the X-list is searched for potential conflict aircraft. If the subject aircraft is on the EX-list, the EX-list is searched. The X-list may also be searched if the aircraft is close to the altitude limit of the X-list or is projected to be within the altitude limit of the X-list within the maximum detection time threshold.

After determining which list the subject aircraft is on, the search limits along that list must be computed. To compute the search limits, the resolution advisory tau threshold and immediate range parameters must be chosen. The maximum values from Table 13-11 are used in the Domino Coarse Screen Filter.

TABLE 13-11

RESOLUTION ADVISORY THRESHOLD
VARIABLES USED IN DOMINO LOGIC

VARIABLE[1]	INDICES		VALUE[3]
	CONTROL STATE	ENAT	
DAF	C/C[2]	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
	C/U	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
	U/U	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
DRCMD2	C/C	1	0.5625 nmi^2
		2	0.5625 nmi^2
		3	0.5625 nmi^2
		4	0.5625 nmi^2
	C/U	1	0.5625 nmi^2
		2	0.5625 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2
	U/U	1	1.0 nmi^2
		2	1.0 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2

TABLE 13-11
(Continued)

PARAMETER	INDICES				VALUE		
	CONTROL STATE	ENAT	MULT	EQUIP	NOMINAL	MIN	MAX
DTCMDH	C/C	1	GT3	E/E[4]	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		2	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		3	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		4	GT3	E/E	TCONH-35	38	38 s
				E/U	TCONH-35	38	38 s
			LE3	E/E	TCONH-35	38	38 s
				E/U	TCONH-35	38	38 s
DTCMDH	C/U	1	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		2	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		3	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		4	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	38	53 s
				U/E	TCONH-15	38	53 s

TABLE 13-11
(Concluded)

PARAMETER	INDICES			VALUE
	CONTROL STATE	ENAT	UUIND[5]	
DTCMDH	U/U	1	1	30 s
			2	38 s
		2	1	30 s
			2	38 s
		3	1	30 s
			2	38 s
		4	1	38 s
			2	38 s
DTCMDV	Same as DTCMDH, except TCONH is replaced by TCONV in the NOMINAL column			

[1]See local variable structure DRAVBL in Section 13.5

[2]C = Controlled U = Uncontrolled

[3]The value of TCONH, TCONV is calculated in Routine TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION.
If the value of DTCMDx after subtracting the offset is not within the bounds specified by the MINimum and MAXimum columns, then the value of DTCMDx is set to the minimum or maximum value specified.

[4]E = ATARS Equipped U = Unequipped

[5]UUIND is defined in Routine DOMINO_UNCON_UNCON_INDEX_DETERMINATION.

The logic that performed the PSEP calculations also saved projected positions and velocities in response to potential resolution advisories in the RAPP Table. To calculate the coarse screen limits, a TCMDH projection is made from each of the points along the response paths in the RAPP Table.

Once the minimum and maximum x and y projected positions have been calculated, a buffer distance (RMAX) must be added to obtain the actual X-list (EX-list) search limits. The buffer accounts for the object aircraft and is equal to the distance that an aircraft going the maximum speed for the X-list (EX-list) can travel during the resolution advisory response projection interval (MANTM + DELAY) and the resolution advisory detection threshold (TCMDH). The maximum speeds are 240 kts (XVEL) for aircraft on the X-list and 600 kts (EXVEL) for aircraft on the EX-list. A maximum vertical maneuver rate of 1000 ft/min (CSCREEN.ZFAST) is assumed for aircraft on the X-list and EX-list.

The altitude limits used in the Domino Coarse Screen Filter are computed similiarly to the horizontal limits. After the search limits have been calculated, the Domino Coarse Screen Filter simply searches along the X-list (EX-list) looking for aircraft that are contained in the x, y and z search limits. Any aircraft within these bounds are added to the Potential Domino Conflict List for the subject aircraft. Any aircraft within the search limits that are already in conflict with and receiving a resolution advisory because of the subject aircraft are not added to the Potential Domino Conflict List.

It is possible for an aircraft to be in more than one conflict on a scan. If this is the case, then it is possible to compute a list of potential domino conflict aircraft twice on the same scan for a particular aircraft. It is best to avoid this duplicate processing if possible. When a list of potential domino conflict aircraft is created for a subject aircraft, a pointer to the head of that list is saved in the Pair Record. Also in the Pair Record is a field of flags indicating which maneuvers were considered in the determination of the list. If an aircraft goes through master resolution and RAER a second time in the same scan, then it is possible that the same Potential Domino Conflict List may be used. If the resolution advisories being considered for the second conflict are the same or a subset of the resolution advisories considered for the first conflict, then the same list of potential domino conflict aircraft may be used. If the other aircraft in the current pair is on the list of potential domino conflict aircraft, it must be deleted from the list.

13.4.2.4.3 Domino Detection Filter

The remainder of the domino logic consists of performing the detection checks for conflicts requiring resolution advisories between the subject aircraft and each of the aircraft on the Potential Domino Conflict List. The detection checks are performed for each aircraft on the Potential Domino Conflict List or until a conflict is found, for each potential resolution advisory for the subject aircraft. The conflict detection parameters are determined from Table 13-11.

13.4.2.4.3.1 Domino Coarse Detection Filter

To minimize the number of times the Domino Detection Filter is performed, a coarse detection check is performed first. The Domino Coarse Detection Filter determines the need for performing the detailed conflict detection checks at each of the projected points. First, it determines if the two aircraft will be in conflict in the vertical dimension at any time during the domino projection interval. If a conflict in the vertical dimension is not possible during this interval, then the detection checks do not have to be performed for this potential domino conflict aircraft. Otherwise, a coarse check in the horizontal dimension is performed. The horizontal check is not passed if the aircraft are diverging and presently separated by more than the immediate range threshold for a conflict. However, if the potential domino conflict aircraft is receiving a positive horizontal resolution advisory, or the potential resolution advisory to the subject aircraft is a positive horizontal resolution advisory, then the horizontal coarse detection check must be performed from all four of the projected positions. All four checks must fail (indicate a conflict is not possible) for the horizontal check to fail.

The Domino Coarse Detection Filter should not be confused with the Domino Coarse Screen Filter. The Domino Coarse Screen Filter generates a list of aircraft which are in the vicinity of the pair of aircraft in conflict. The Domino Coarse Detection Filter is a way to reduce the computations needed to determine conflicts between aircraft on the Potential Domino Conflict List and a subject aircraft from the conflict pair.

13.4.2.4.3.2 Domino Resolution Advisory Detection Filter

The Domino Resolution Advisory Detection Filter performs only those checks necessary to detect a conflict requiring resolution advisories. Those checks include the tau and immediate range checks in both the horizontal and vertical dimensions. The maneuvering target threat check is not performed.

If the non-mode C option of the domino logic is being performed, then some of the potential domino conflict aircraft may not have mode C data. In a pair including a non-mode C aircraft, the aircraft are assumed to be in conflict in the vertical dimension.

The checks for determining the need for resolution advisories require the computation of tau values and immediate range values. The domino detection filter must compute these values at four points along the projected path of each potential domino conflict aircraft paired with an aircraft from the subject pair. In addition, it is possible to repeat the same calculations twice between the same two aircraft. For example, if a maneuvered aircraft may receive either a turn left or turn right advisory, then the domino logic would repeat the vertical tau and separation calculations against a particular aircraft when checking each horizontal advisory for causing a conflict. The number of computations could be reduced by remembering the outcome of the detection checks between the two aircraft. The data structure for the Potential Domino Conflict List contains fields to remember the outcome of the detection checks.

13.4.3 Tie-Breaking Feature

It is possible for two or more PRA sets to have the same features set after all absolute and relative features are evaluated. In case of a tie, a final tie-breaking feature is used to break the tie. This feature checks for the largest predicted separation after the aircraft respond to the advisories. If a tie still remains, predicted separation shortly after the aircraft begin to respond to the advisories (at QTIME) is checked using the QSEP matrix. This feature is designed to always break ties.

13.5 Pseudocode for Resolution Advisories Evaluation Routine

The high and low level pseudocode for the Resolution Advisories Evaluation Routine is included in this section. Most notation conventions are described in Section 12.4.

Because of the complexity of this logic, the lower level processes are not all simple routines. In some cases, they are themselves high level logical routines that have more than one lower level of subprocess. Tables 13-12 through 13-15 show the process breakdown of RAER.

TABLE 13-12

RESOLUTION ADVISORIES EVALUATION
ROUTINE PROCESS HIERARCHY

1. maneuvering_AC_determination[1]

2. potential_resolution_advisory_sets_selection

   o two_AC_resolution_logic_vertical_resolution_advisories_selection

3. PSEP_MATRIX_GENERATOR[2]

4. potential_resolution_advisory_sets_culling

   o negative_resolution_advisory_determination
      - vertical_divergence_logic
      - one_AC_maneuvering_negative_vertical_resolution_advisory_test
      - positive_to_negative_resolution_advisory_conversion

   o VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION
      - converging_AC_check
      - vertical_speed_limit_calculation

   o absolute_features_evaluation_two AC_resolution definition[3]

5. multi_AC_resolution

   o resolution_advisory_compatibility_with_existing_conflicts
      - feature_maneuvered_unmaneuvered_conflict_multi_AC_definition

   o multi_AC_conflict_possible_resolution_advisories
      - PSEP_MATRIX_GENERATOR[2]

   o multi_AC_resolution_logic_advisories_calculations
      - absolute_features_evaluation_multi_AC_resolution_definition
         -- feature_deliverable
         -- feature_dimension_available
         -- feature_maneuvered_unmaneuvered_conflict_multi_AC_definition
            - multi_AC_vertical_maneuvered_unmanuevered_conflict_determination RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION[4]
            - multi_AC_horizontal_maneuvered_unmaneuvered_conflict_determination RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION[4]

TABLE 13-12
(Concluded)

6. final_resolution_advisory_selection
    - o relative_features_evaluation[3]
    - o highest_valued_potential_resolution_advisory_sets_count
    - o feature_domino_logic[3]
    - o tie_breaker_features_evaluation[3]
    - o PSEP_model_validation_values_saved
7. PSEP_model_validation_values_saved

[1]The capitalization in Tables 13-12 thru 13-14 conforms to that used in the pseudocode. Process names are in lower case and Task and Routine names are in upper case.
[2]This process is presented in detail in Table 13-13.
[3]This process is presented in detail in Table 13-14.
[4]This routine is presented in detail in Table 13-15.

TABLE 13-13

PSEP MATRIX GENERATOR ROUTINE HIERARCHY

```
1.  PSEP_MATRIX_GENERATOR

        •  modeling_of_delay_period
              -  linear_modeling_of_delay
                   --  VERTICAL_ADVANCEMENT
                   --  CONTINUE_STRAIGHT
                   --  CONVERGENCE_3D
                   --  MISS_DISTANCE_3D
                   --  CONVERGENCE_HORIZONTAL
                   --  MISS_DISTANCE_HORIZONTAL
              -  nonlinear_modeling_of_delay
                   --  nonlinear_delay_preparations
                         - FINAL_VERTICAL_RATE_DETERMINATION
                         - COMPUTATION_OF_TURN_CONSTANTS
                   --  nonlinear_advancement
                         - VERTICAL_ADVANCEMENT
                         - TURN_LEFT
                         - TURN_RIGHT
                         - CONTINUE_STRAIGHT

        •  vertical_level_selection
              -  vertical_rate_determination

        •  horizontal_path_selection

        •  maneuver_time_calculation

        •  maneuver_modeling
              -  geometry_initialization
              -  COMPUTATION_OF_TURN_CONSTANTS
              -  incremental_advancement
                   --  VERTICAL_ADVANCEMENT
                   --  CONTINUE_STRAIGHT
                   --  TURN_LEFT
                   --  TURN_RIGHT
                   --  addition_to_RAPP_table
              -  separation_calculations
              -  collection_of_minimums

        •  vertical_convergence_checks
```

TABLE 13-13
(Concluded)

- horizontal_convergence_checks
  - CONVERCENCE_HORIZONTAL
  - MISS_DISTANCE_HORIZONTAL
- three_dimensional_convergence_checks
  - CONVERGENCE_3D
  - MISS_DISTANCE_3D

TABLE 13-14

RAER FEATURES HIERARCHY

1. absolute_features_evaluation_two_AC_resolution_definition

   - feature_deliverable
   - feature_dimension_available
   - feature_maneuvered_unmaneuvered_conflict_two_AC_definition
     - two_AC_vertical_maneuvered_unmaneuvered_conflict_determination
     - two_AC_horizontal_maneuvered_unmaneuvered_conflict_determination

2. relative_features_evaluation

   - feature_PSEP_GE_SEP1
   - feature_reinforce_res_adv_from_non_connected_site_or_BCAS
     - other_sources_resolution_advisory_determination
   - feature_terrain_or_obstacle_alert
   - feature_aircraft_far_from_radar
   - feature_negative_resolution_advisories_suffice
   - feature_negative_resolution_advisories_do_not_reverse_maneuver
   - feature_fast_unmaneuvered_slow_maneuvered
   - feature_unmaneuvered_with_large_vertical_rate
   - feature_no_level_off_time_for_verticals
   - feature_non_response_to_positive_resolution_advisories_detected
   - feature_aircraft_on_final_approach
   - feature_initial_resolution_advisory_selection
   - feature_PSEP_GE_SEP2
   - feature_compatible_with_turn

TABLE 13-14
(Continued)

```
        o  feature_big_vertical_miss_distance

        o  feature_big_horizontal_miss_distance

        o  same_weight_calculations

        o  feature_reinforces_prior_resolution_advisories

        o  feature_speed_check

        o  feature_reinforces_turn

3.  feature_domino_logic

        o  domino_conflict_detection
             -  potential_domino_conflict_list_creation
                  --  pair_record_check_for_existing_potential_domino_conflict_list
                        - potential_domino_conflict_list_copy
                            potential_domino_conflict_list_entry_addition

                  --  domino_coarse_screen
                        - potential_resolution_advisory_status_variable_determination
                        - domino_search_area_subject_AC_calculations
                            domino_search_area_horizontal_dimension_calculations
                            NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING
                                VERTICAL_ADVANCEMENT
                              vertical_only_nonlinear_modeling_of_delay
                                FINAL_VERTICAL_RATE_DETERMINATION
                              vertical_only_modeling_of_maneuver_period
                                VERTICAL_ADVANCEMENT
                            domino_search_area_vertical_dimension_calculations

                        - EX_list_object_AC_domino_buffer_area_calculations
                        - EX_list_domino_search_limits_calculations
                        - EX_list_domino_search
                            EX_list_forward_domino_search
                              domino_coarse_screen_altitude_conflict_test
                                potential_domino_conflict_list_entry_addition
                            EX_list_backward_domino_search
                              domino_coarse_screen_altitude_conflict_test
                                potential_domino_conflict_list_entry_addition
```

TABLE 13-14
(Concluded)

```
                    - X_list_object_AC_domino_buffer_area_calculations
                    - X_list_domino_search_limits_calculations
                    - X_LIST_SIGNPOST_ENTRY_CALCULATION
                    - X_list_domino_search
                         X_list_forward_domino_search
                           domino_coarse_screen_altitude_conflict_test
                             potential_domino_conflict_list_entry_addition
                         X_list_backward_domino_search
                           domino_coarse_screen_altitude_conflict_test
                             potential_domino_conflict_list_entry_addition

        -  domino_resolution_advisory_detection_filter
             --  domino_detection_thresholds
                    - domino_encounter_AC_multiplicity_determination
                    - ENCOUNTER_AREA_TYPE_DETERMINATION
                         FINAL_APPROACH_ZONE_DETERMINATION
                         AREA_TYPE_DETERMINATION
                    - AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION
                    - DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
                         DOMINO_UNCON_UNCON_INDEX_DETERMINATION
             --     domino_coarse_detection_checks
                    - domino_course_detection_vertical_dimension_checks
                    - domino_course_detection_horizontal_dimension_checks
             --     DOMINO_RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
             --     non_mode_C_resolution_tau_and_proximity_comparisons

    o  domino_features_weight_addition

4.  tie_breaker_features_evaluation

    o  feature_biggest_separation_for_negatives

    o  feature_biggest_separation_for_positives
```

TABLE 13-15

RESOLUTION ADVISORY MODELING FOR
PREDICTED SEPARATION ROUTINE HIERARCHY

```
1. RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION

   o one_path_modeling_of_delay_period
     - one_path_linear_modeling_of_delay
       -- VERTICAL_ADVANCEMENT
       -- CONTINUE_STRAIGHT
       -- CONVERGENCE_3D
       -- MISS_DISTANCE_3D

     - one_path_nonlinear_modeling_of_delay
       -- FINAL_VERTICAL_RATE_DETERMINATION
       -- COMPUTATION_OF_TURN_CONSTANTS
       -- nonlinear_advancement
          - VERTICAL_ADVANCEMENT
          - TURN_LEFT
          - TURN_RIGHT
          - CONTINUE_STRAIGHT

   o maneuver_time_calculation
   o FINAL_VERTICAL_RATE_DETERMINATION
   o one_path_maneuver_modeling
     - COMPUTATION_OF_TURN_CONSTANTS
     - one_path_incremental_advancement
       - VERTICAL_ADVANCEMENT
       - TURN_LEFT
       - TURN_RIGHT
       - CONTINUE_STRAIGHT

   o one_path_3D_convergence_check
     - CONVERGENCE_3D
     - MISS_DISTANCE_3D
```

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

STRUCTURE RAERPARM


  GROUP  domino

     INT EXVEL          <max velocity of AC on EX-list>

     FLT MAXAP          <max immediate altitude separation threshold>

     FLT MAXTLI         <max controlled AC tau detect threshold for Domino
                          Detection>

     FLT MAXTLV         <max uncontrolled AC tau detect threshold for Domino
                          Detection>

     INT XVEL           <max velocity of AC on X-list>


  GROUP pointers

     PTR BACMRADS       <pointer to RADS with both AC maneuvered>

     PTR FACMRADS       <pointer to RADS with first AC maneuvered>

     PTR MOPVRADS       <pointer to RADS for multi-AC resolution logic
                          with opposite sense vertical RAs to each AC>

     PTR MSMVRADS       <pointer to RADS for multi-AC resolution logic with
                          same sense vertical RAs to each AC>

     PTR SACMRADS       <pointer to RADS with second AC maneuvered>


  GROUP negative_RA

     FLT ATERN          <altitude above terrain, below which a Descend must
                          be changed to a Don't Climb>

     FLT MRATE          <minimum vertical rate required for an AC before a VSL
                          may be chosen for that AC>

     FLT NSVDAT         <vertical divergence altitude threshold for negative
                          RA suffices determination logic>

     FLT NSVDPT         <vertical divergence projection time for negative RA
                          suffices determination logic>


  GROUP multi-AC

     FLT ZCARE          <altitude separation, below which opposite sense vertical
                          RA's are selected; above which same sense vertical RA's
                          are selected>

-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL PARAMETERS  -------------
```

```
----------------------------------------------------------------------------

GROUP feature_weights
   INT BIGHWGT          <weight for big horizontal miss distance feature>
   INT BIGVWGT          <weight for big vertical miss distance feature>
   INT BSEPNWGT         <weight for big separation for negatives>
   INT BSEPPWGT         <weight for big separation for positives>
   INT COMWTWGT         <weight for compatible with turn>
   INT DELWGT           <weight for deliverable feature>
   INT DIMAVWGT         <weight for dimension available feature>
   INT DOM1WGT          <weight for one AC caused a domino conflict feature>
   INT FARRAWGT         <weight for far from radar feature>
   INT FAZWGT           <weight for AC on final approach>
   INT FUCSCWGT         <weight for fast uncmded/slow cmded>
   INT NDNRMWGT         <weight for negative does not reverse maneuver>
   INT NDOMWGT          <weight for neither AC caused a domino conflict feature>
   INT NEGSFWGT         <weight for negative suffices>
   INT NOLEVWGT         <weight for no level-off time for verticals>
   INT NRESPWGT         <weight for non-response to positive res adv detected>
   INT OTHSTWGT         <weight for reinforce other sites feature>
   INT PSEP1WGT         <weight for PSEP1 GE SEP1 feature>
   INT PSEP2WGT         <weight for PSEP2 GE SEP2 feature>
   INT REINTWGT         <weight for reinforces turn>
   INT REPRAWGT         <weight for reinforce prior res adv>
   INT SNGLDWGT         <weight for logic favoring single dimension res adv>
   INT SPDCKWGT         <weight for speed check>
   INT TEROBWGT         <weight for terrain or obstacle alert>
   INT UCLVRWGT         <weight for uncmded with large vertical rate>
   INT UNMANWGT         <weight for maneuvered/unmaneuvered conflict feature>
```

```
--------------------------------------------------------------------------------

  GROUP features
     FLT MDHMSQ      <horizontal miss distance squared, above which res adv sets with
                       horiz components are favored, when at least one AC is turning>
     FLT MDHSQ       <horizontal miss distance squared, above which res adv sets with
                       horiz components are favored, when neither AC is turning>
     FLT RDISTR      <distance from radar where range-azimuth data becomes unreliable>
     FLT SEP2AP      <SEP2 threshold percentage>
     FLT TV1         <time threshold to allow vertical crossover>
     FLT TV2         <time threshold to allow vertival level-off>
     FLT TXTH1       <lower limit of track crossing angle for which double
                       dimension res adv are prefered>
     FLT TXTH2       <upper limit of track crossing angle for which double
                       dimension res adv are preferred>
     FLT VFASTSQ     <horiz speed for maneuvering AC above which vert
                       or double dimension res adv are preferred>
     FLT VRATIO      <ratio of squared speed of uncmded AC to cmded AC,
                       above which double dimension res adv are preferred>
     FLT VSLOWSQ     <horiz speed for maneuvering AC below which
                       horiz or double dimension res adv are preferred>
     FLT ZDTH        <vert rate used to determine when an uncmded AC has a
                       threatening rate.  Also, used to determine when neg vert res
                       adv would be disruptive to a maneuvered AC>

  GROUP misc
     FLT DOMCRSE     <scans of domino coarse detection checks>
     FLT DOMSCANS    <scans of domino projection interval after DELAY period>
     FLT DOMSRCH     <scans of domino coarse detection checks>
     FLT TVRULE      <projection time for 'eight second rule'>

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

STRUCTURE LOGIC_TABLES

  GROUP compatible_res_adv
     BIT COMPAT(11,11)          <table of resolution advisory compatibility:
                                  COMPAT(new res adv, previous res adv);
                                  true when compatible>
  GROUP reinf_res_adv
     BIT REINF(9,11)            <table of resolution advisory reinforcement;
                                  REINF(new res adv, previous res adv);
                                  true when new reinforces old>
  GROUP compat_turn_states
     BIT COMPATTS(7,6)          <table of horiz turn status compatibility with
                                  res adv:  COMPATTS(turn status, res adv);
                                  true when compatible>
     BIT COMPATZD(3,3)          <table of vert velocity compatibility with
                                  res adv:  COMPATZD(ACID.ZD, res adv)
                                  true when compatible>

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------------------

<*** PARAMETERS USED IN PSEP MODELING OF AIRCRAFT ***>

STRUCTURE MODELING

  GROUP values
     FLT ACCELC              < Upward acceleration rate >
     FLT ACCELD              < Downward acceleration rate >
     FLT BANKA               < Assumed bank angle for all turns >
     FLT DELAY               < Length of modeling delay period >
     FLT DELINT              < Time interval for nonlinear modeling of delay period >
     FLT G                   < Acceleration due to gravity >
     FLT MTLL                < Lower limit on maneuver time >
     FLT MTSC                < Slow-closing value for maneuver time >
     FLT MTUL                < Upper limit on maneuver time >
     FLT QTIME               < Projection time for saving QSEP matrix >
     FLT TCADEL              < Time increment for computing maneuver time >
     FLT TIMINT              < Time interval for maneuver modeling >
     FLT TNVRAM              < Maneuver time for modeling negative vertical RAs >
     FLT TURNA1              < Maximum turn angle for one aircraft >
     FLT TURNA2              < Maximum combined turn angle for two aircraft >
     FLT VRTH2               < Slow-closing velocity threshold >
     FLT VTHSQ               < Threshold for 'fast' vs. 'slow' aircraft >
     FLT V1000               < Vertical rate modeled for 1000 ft/min VSL >
     FLT V2000               < Vertical rate modeled for 2000 ft/min VSL >
     FLT V500                < Vertical rate modeled for 500 ft/min VSL >
     FLT ZDDWNF              < Achievable descent rate for a 'fast' aircraft >
     FLT ZDDWNS              < Achievable descent rate for a 'slow' aircraft >
     FLT ZDUPF               < Achievable climb rate for a 'fast' aircraft >
     FLT ZDUPS               < Achievable climb rate for a 'slow' aircraft >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

STRUCTURE RAERVBL

  GROUP logic_path
     BIT MRNCAP        <Master Resolution called RAER when true, Conflict Resolution
                         Data Task called RAER when false>
     BIT SNGDIM        <single dimension RA's preferred when true, double dim, when
                         false>

  GROUP res_adv
     FLT ASEP          <altitude separation threshold>
     BIT FSTUNCZD      <set true when ZD of uncmded AC is threatening>
     FLT MDHM          <threshold for favoring res adv with a horiz component when
                         large horizontal miss distance is projected>
     INT NPRAABS       <number of RADS with all absolute features favored>
     INT OSHMAN1       <horizontal RA from other site or BCAS>
     INT OSHMAN2       <horizontal RA from other site or BCAS>
     INT OSVMAN1       <vertical RA from other site or BCAS>
     INT OSVMAN2       <vertical RA from other site of BCAS>
     BIT RASELECT      <res adv selected this scan for this pair>
     BIT RSPND1        <AC is to be maneuvered by RAEL, if true>
     BIT RSPND2        <AC is to be maneuvered by RAEL, if true>
     FLT TRATIO        <speed ratio of uncmded AC to cmded AC>
     FLT TVALUE        <temporary value of features>
     FLT TVERT         <temporary vertical resolution advisory>
     FLT TXTH          <track crossing angle>
     INT VERTRA1       <vert res adv selected by 8 sec rule for 1st AC of pair>
     INT VERTRA2       <vert res adv selected by 8 sec rule for 2nd AC of pair>
     INT Z8SEC1        <8 sec altitude projection from current time for first AC>
     INT Z8SEC2        <8 sec altitude projection from current time for second AC>
```

```
--------------------------------------------------------------------------------

 GROUP pointers
    PTR ELENTRY           <encounter list entry>
    PTR PREC              <pair record pointer>
    PTR RADS              <pointer to list of potential RA's for the current pair>
    PTR RADSPTR           <pointer to selected res adv>
    PTR RAPP1             <pointer to RAPP Table for first AC in pair>
    PTR RAPP2             <pointer to RAPP Table for second AC in pair>
    PTR TRADS             <temporary RADS pointer>

 GROUP neg_res_adv
    INT HDTHR             <sq of neg hor res adv thr, may be modified value in some
                             cases>
    INT NEGDIV            <neg vert res adv divergence threshold>
    FLT TRTHO             <true horizontal tau>

ENDSTRUCTURE;
```

```
-----------------------------------------------------------------------------------------

STRUCTURE DOMINOVBL

  GROUP coarse_screen
     PTR NXTAC         <object AC being examined on X-EX-list by domino logic>
     FLT RMAX          <max horiz range traversed by intruder during domino interval>
     FLT TLD           <max detection threshold>
     FLT XL            <lower limit of X direction search>
     FLT XMAX          <subject AC max X during domino search interval>
     FLT XMIN          <subject AC min X during domino search interval>
     FLT XPR(9)        <domino coarse screen projection of subject AC>
     FLT XU            <upper limit of X direction search>
     FLT YL            <lower limit of Y direction search>
     FLT YMAX          <subject AC max Y during domino search interval>
     FLT YMIN          <subject AC min Y during domino search interval>
     FLT YPR(9)        <domino coarse screen projections of subject AC>
     FLT YU            <upper limit of Y direction search>
     FLT ZL            <lower limit of Z direction search>
     FLT ZMAX          <upper vertical extent of subject AC during domino
                          search interval>
     FLT ZMIN          <lower vertical extent of subject AC during domino
                          search interval>
     FLT ZPR(5)        <domino coarse screen projections of subject AC>
     FLT ZU            <upper limit of Z direction search>
```

```
--------------------------------------------------------------------------------------

  GROUP detection
    FLT AVRZ          <absolute value of relative vertical velocity>
    FLT DALT          <current altitude separation>
    BIT DCMDFLG       <CMDFLG for domino detection logic>
    FLT DDOT(4)       <domino detection DOT values>
    FLT DDSQ          <modified horizontal tau distance>
    INT DENAT         <encounter area type used in domino logic>
    FLT DRANGE2       <range>
    FLT DRZ           <vertical velocity difference>
    FLT DTH           <horizontal tau>
    FLT DTV           <vertical tau>
    FLT DVRZ          <vertical velocity difference>
    INT INFAZ2        <number of AC in final approach zone>
    INT MAXVALUE      <value of features for RADS with all absolute features favored>
    INT NUMPRA        <number of RADS tied with highest value of features favored>
    INT PREQ          <number of AC to be maneuvered in domino pair>
    INT PRCONT        <controlled status for domino pair>
    INT ZMX           <max vertical distance traversed by intruder at
                         velocity CSCREEN.ZFAST during domino interval>

ENDSTRUCTURE;

STRUCTURE DRAVLB
  GROUP thresholds
    FLT DAF           <current altitude separation threshold for giving RAs>
    FLT DRCMD2        <current range separation threshold for giving RAs>
    FLT DTCMDH        <horizontal tau threshold for giving RAs>
    FLT DTCMDV        <vertical tau threshold for giving RAs>
ENDSTRUCTURE;
```

```
---------------------------------------------------------------------------

STRUCTURE RADS                <Resolution Advisory Data Structure>
                              <Refer to Table 13-4 in text for initial values>
                              <40 of these data structures are needed>

  GROUP pointers
    PTR NXTADV                 <next RADS in list>

  GROUP advisory_components
    INT H1                     <horizontal component of AC 1's res adv>
    INT H2                     <horizontal component of AC 2's res adv>
    INT V1                     <vertical component of AC 1's res adv>
    INT V2                     <vertical component of AC 2's res adv>

  GROUP read-only_flags
    BIT CMDED_CMDED            <advisory set maneuvers both AC>
    BIT CMDED_UNCMDED          <advisory set maneuvers first AC in pair>
    BIT HORIZ                  <advisory in horiz dimension when true>
    BIT SINGLE                 <advisory is one dimension only when true>
    BIT UNCMDED_CMDED          <advisory maneuvers second AC in pair>
    BIT VERT                   <advisoy in vertical dimension when true>

  GROUP read/write_flags
    BIT BELOW1000              <descend res adv must be changed to negative>
    BIT NEGATIVE               <negative res adv provide sufficient separation>

  GROUP sep_matrix_indices
    INT INDEX1                 <PSEP & HMD index for first AC's horizontal advisory>
    INT INDEX2                 <PSEP & HMD index for second AC's horizontal advisory>
    INT INDEX3                 <PSEP & VMD index for vertical level>
    PTR MATPTR                 <pointer to separation matrices to be used with
                                 this resolution advisory set>



------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL VARIABLES ------------
```

```
-------------------------------------------------------------------------------

 GROUP other-info
   INT DOMVALUE              <computed value of this advisory's features down to
                               domino features>
   BIT FEATBITS(25)          <one bit for each of 25 features>
   INT VALUE                 <computed relative value of this advisory's features>

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

STRUCTURE RAPP_TABLE  <Resolution Advisory Projected Position Table>

  GROUP positions
     FLT X(3,4)              <(maneuver, scan)
                               (CS, TL, TR;
                               CTIME + DELAY + SCANT, CTIME + DELAY + 2 * SCANT,
                               CTIME + DELAY + 3 * SCANT, CTIME + DELAY + 4 * SCANT)>
     FLT Y(3,4)
     FLT Z(5,4)              <(CVV, CL, DES, DDES or LDES, DCL or LCL;
                               CTIME + DELAY + SCANT, CTIME + 2 * SCANT,
                               CTIME + DELAY + 3 * SCANT, CTIME + DELAY + 4 * SCANT)>

  GROUP velocities
     FLT XD(3,4)             <(maneuver, scan)>
     FLT YD(3,4)
     FLT ZD(5,4)

ENDSTRUCTURE;


<*** PREDICTED SEPARATION MATRICES ***>

STRUCTURE PSMAT              < may need two structures for each AC in the
                                subject conflict pair>

   GROUP minimums
     FLT HMD2(3,3)           < Horizontal miss distances squared >
     FLT PSEP2(3,3,3)        < 3-D miss distances squared, vertical weighted >
     FLT VMDA(3)             < Vertical minimum separation values >
     FLT VMDB(3)             < Vertical components of PSEP2 for 'straight' paths >

   GROUP snapshot
     FLT QSEP2(3,3,3)        < 3-D separation values QTIME seconds after delay >

ENDSTRUCTURE;
------------  RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL VARIABLES  -------------
```

```
--------------------------------------------------------------------------------

STRUCTURE PDC_LIST          <Potential Domino Conflict list element>
                            <one entry for each domino object AC>

  GROUP pointer
     PTR INTRAC             <pointer to state vector of this PDC aircraft>
     PTR NXTINTR            <pointer to next element in this PDC list>

  GROUP res_adv             <three states: $NOTTST, $NODOMC, $DOMC>
     INT CLIMB              <climb res adv>
     INT DCLIMB             <don't climb or limit climb res adv>
     INT DDESC              <don't descend or limit descend res adv>
     INT DESC               <descend res adv>
     INT DLEFTDRIGHT        <don't turn left and/or don't turn right>
     INT LEFT               <turn left res adv>
     INT LEFTCLIMB          <left, climb advisory>
     INT LEFTDESC           <left, descend advisory>
     INT RIGHT              <turn right res adv>
     INT RIGHTCLIMB         <right, climb advisory>
     INT RIGHTDESC          <right, descend advisory>

  GROUP detection
     INT ENAT               <area type this encounter is in based on
                               current position of both AC>
     INT MULT               <number of AC in conflict cluster, including this AC>

ENDSTRUCTURE;
```

```
------------------------------------------------------------------------------

STRUCTURE PRADSVVBL      <Potential Resolution Advisory Domino Status Variables
                               for subject AC>

  GROUP ac1               <four possible states: $NOPRA, $DOMNP, $DOMCNC, $DOMCC>
     INT CLMB             <climb res adv>
     INT DSC              <descend res adv>
     INT LFT              <turn left res adv >
     INT LFTCLMB          <turn left, climb res adv>
     INT LFTDSC           <turn left, descend res adv>
     INT NCLMB            <don't climb or limit climb res adv>
     INT NDSC             <don't descend or limit descend res adv>
     INT NLFTNRGT         <don't turn left or don't turn right>
     INT RGT              <turn right res adv>
     INT RGTCLMB          <turn right, climb res adv>
     INT RGTDSC           <turn right, descend res adv>

  GROUP ac2
     LIKE PRADSVVBL.ac1

ENDSTRUCTURE;
```

```
------------------------------------------------------------------------

<*** VARIABLES USED IN PSEP MODELING OF AIRCRAFT ***>

STRUCTURE MODVBL

  GROUP miscellaneous
     FLT MANTM          < Maneuver time >

  GROUP relative_geometry
     FLT RX             < Relative X-position >
     FLT RY             < Relative Y-position >
     FLT RZ             < Relative Z-position >
     FLT VRX            < Relative X-velocity >
     FLT VRY            < Relative Y-velocity >
     FLT VRZ            < Relative Z-velocity >
     FLT VR2            < Magnitude squared of relative velocity >
     FLT DOT            < Separation * rate-of-change of separation >

ENDSTRUCTURE;


STRUCTURE RATE

  GROUP ac1
     FLT CLM            < 'Climb' rate to be achieved >
     FLT DES            < 'Descend' rate to be achieved >
     FLT DCLM           < 'Don't climb' rate to be achieved >
     FLT DDES           < 'Don't descend' rate to be achieved >
     FLT ZDFD           < Final rate to be achieved during delay period >
     FLT ZDFM(3)        < Final rate for each level during maneuver period >

  GROUP ac2
     LIKE RATE.ac1

ENDSTRUCTURE;

------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL VARIABLES -------------
```

```
--------------------------------------------------------------------------------------

<*** VARIABLES USED IN PSEP MODELING OF AIRCRAFT ***>

STRUCTURE PATH

  GROUP ac1
     BIT MODEL(3)        < Indicates whether each path will be modeled >

  GROUP ac2
     LIKE PATH.ac1

ENDSTRUCTURE;

STRUCTURE TURCON         < Turn constants >

  GROUP ac1
     FLT SA
     FLT CA
     FLT A
     FLT B

  GROUP ac2
     LIKE TURCON.ac1

ENDSTRUCTURE;

STRUCTURE PREVIOUS

  GROUP advisories
     INT PHRA1          < Previous horizontal RA for aircraft 1 >
     INT PVRA1          < Previous vertical RA for aircraft 1 >
     INT PHRA2          < Previous horizontal RA for aircraft 2 >
     INT PVRA2          < Previous vertical RA for aircraft 2 >

ENDSTRUCTURE;

------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL VARIABLES ------------
```

```
--------------------------------------------------------------------------------
<*** VARIABLES USED IN PSEP MODELING OF DELAY PERIOD ***>

STRUCTURE DELGEOM       < Delay geometry >

  GROUP hor1            < Horizontal for aircraft 1 >
     FLT X
     FLT Y
     FLT XD
     FLT YD

  GROUP hor2            < Horizontal for aircraft 2 >
     LIKE DELGEOM.hor1

  GROUP ver1            < Vertical for aircraft 1 >
     FLT Z
     FLT ZD

  GROUP ver2            < Vertical for aircraft 2 >
     LIKE DELGEOM.ver1

  GROUP minsep
     FLT PSEP2I         < Initial value for PSEP2 >
     FLT HMD2I          < Initial value for HMD2 >
     FLT VMDAI          < Initial value for VMDA >
     FLT VMDBI          < Initial value for VMDB >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

<*** VARIABLES USED IN PSEP MODELING OF MANEUVER PERIOD ***>

STRUCTURE MANGEOM        < Maneuver geometry >

  GROUP hor1(3)          < Horizontal paths for aircraft 1 >
     FLT X
     FLT Y
     FLT XD
     FLT YD

  GROUP hor2(3)          < Horizontal paths for aircraft 2 >
     LIKE MANGEOM.hor1

  GROUP ver1(3)          < Vertical levels for aircraft 1 >
     FLT Z
     FLT ZD

  GROUP ver2(3)          < Vertical levels for aircraft 2 >
     LIKE MANGEOM.ver1

  GROUP separation       < Current separation matrices >
     FLT CURP2(3,3,3)
     FLT CURH2(3,3)
     FLT CURV(3)

ENDSTRUCTURE;
```

------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOCAL VARIABLES ------------

```
-----------------------------------------------------------------------------------

<*** VARIABLES USED IN NEGATIVE VERTICAL RESOLUTION ADVISORY MODELING ***>

STRUCTURE NVGEOM

  GROUP ver
     FLT Z               < Modeled altitude of aircraft >
     FLT ZD              < Modeled vertical rate of aircraft >

  GROUP prevert
     INT VRAP            < Previous vertical RA for aircraft >

ENDSTRUCTURE;
```

```
-------------------------------------------------------------------------------------
ROUTINE RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
   IN (encounter list entry, pair record, altitude separation threshold,
         single/double dimension flag, calling routine flag)
   OUT (pointer to selected set of resolution advisories);

     <Select a set of resolution advisories for a conflict pair, identified
      in an encounter list entry.  This is done by first determining a list
      of potential resolution advisories based on which aircraft are maneuvered.
      One set of advisories is chosen to resolve the pair from the list of
      potential advisories.  Each advisory set is modeled and the predicted
      separation is calculated.  The resolution advisory set chosen must meet
      certain minimum criteria to be selected.  If no advisory sets meet
      the minimum criteria, then a null pointer is returned to the calling
      routine.

      This routine is refered to as RAER throughout the pseudocode.>

     PERFORM maneuvering_AC_determination;
     PERFORM potential_resolution_advisory_sets_selection;
     CLEAR all fields in two Resolution Advisory Projected Position tables;
     CALL PSEP_MATRIX_GENERATOR;
     PERFORM potential_resolution_advisory_sets_culling;

     IF (no potential resolution advisory sets have all absolute features set)
          THEN PERFORM multi_AC_resolution;
          ELSE;

     IF (more than one resolution advisory set has all absolute features set)
          THEN PERFORM final_resolution_advisory_selection;
     ELSEIF (one resolution advisory set has all absolute features set)
          THEN PERFORM PSEP_model_validation_values_saved;
     OTHERWISE;      <not able to select RADS for the conflict pair>

END RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;

-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  ------------
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------
ROUTINE RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
   IN (FLENTRY, PREC, ASEP, SNGDIM, MRNCAP)
   OUT (RADSPTR);

     PERFORM maneuvering_AC_determination;
     PERFORM potential_resolution_advisory_sets_selection;

     CLEAR all fields in RAPP tables pointed to by RAPP1 and RAPP2;
     CALL PSEP_MATRIX_GENERATOR
             IN (ACID1, ACID2, RSPND1, RSPND2, VERTRA1, VEPTRA2)
             OUT (RADS.MATPTR, RAPP1, RAPP2);
     PERFORM potential_resolution_advisory_sets_culling;

     IF (NPRAABS EQ 0)
          THEN PERFORM multi_AC_resolution;
          ELSE;

     IF (NPRAABS GT 1)
          THEN PERFORM final_resolution_advisory_selection;
     ELSEIF (NPRAABS EQ 1);
          THEN PERFORM PSEP_model_validation_values_saved;
     OTHERWISE;     <not able to select RADS for the conflict pair>

END RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;
```

```
--------------------------------------------------------------------------------
PROCESS maneuvering_AC_determination;

          <This process determines which aircraft should receive a
           resolution advisory (be maneuvered).>

     CLEAR maneuvering flags;
     LOOP:
          Get next AC of subject pair;
     EXITIF (done both aircraft);
          IF (RAER called from Master Resolution Task)
               THEN IF ((AC is ATARS equipped) AND
                              ((AC is uncontrolled) OR (pair record flag
                              indicates controlled AC needs a resolution advisory)))
                         THEN SET maneuvering flag;
                         ELSE;

               <A pair record may not exist if RAER is called from Conflict
                Resolution Data Task.>
               ELSE IF ((AC is ATARS equipped) AND ((AC is uncontrolled) OR
                              (detection flag indicates controlled AC should
                              receive a resolution advisory) OR (either AC
                              is in a conflict receiving resolution advisories)))
                         THEN SET maneuvering flag;
                         ELSE;
     ENDLOOP;
     IF (first AC is in final approach zone)
          THEN IF (second AC is maneuvered)
                    THEN CLEAR maneuvering flag for first AC;
                    ELSE;
     ELSEIF (second AC is in final approach zone)
          THEN IF (first AC is maneuvered)
                    THEN CLEAR maneuvering flag for second AC;
                    ELSE;
     OTHERWISE;
END maneuvering_AC_determination;
```

```
------------------------------------------------------------------------------------

PROCESS maneuvering_AC_determination;

     RSPND1 = $FALSE;
     RSPND2 = $FALSE;

     LOOP;
          Get next AC of subject pair;
     EXITIF (both AC processed);
          IF (MRNCAP EQ $TRUE)
               THEN IF ((ACID.ATSEQ NE $UNEQ) AND
                              ((ACID.CUNC EQ $FALSE) OR (PREC.PIFR EQ $TRUE)))
                         THEN RSPND = $TRUE;
                         ELSE;
               ELSE IF ((ACID.ATSEQ NE $UNEQ) AND
                              ((ACID.CUNC EQ $FALSE) OR
                              ((ELENTRY.IFRFLG EQ $TRUE) OR
                              (ACID1.CTPTR NE $NULL) OR (ACID2.CTPTR NE $NULL))))
                         THEN RSPND = $TRUE;
                         ELSE;
     ENDLOOP;

     IF (ACID1.FAZ NE $FAZ0)
          THEN IF (RSPND2 EQ $TRUE)
                    THEN RSPND1 = $FALSE;
                    ELSE;
     ELSEIF (ACID2.FAZ NE $FAZ0)
          THEN IF (RSPND1 EQ $TRUE)
                    THEN RSPND2 = $FALSE;
                    ELSE;
     OTHERWISE;

END maneuvering_AC_determination;
```

```
--------------------------------------------------------------------------------------

PROCESS potential_resolution_advisory_sets_selection;

    <Select the set of potential resolution advisories from the master
     list of RADS.  The potential set of advisories maneuvers only the first AC,
     second AC, or both AC as indicated by the maneuvering flags.  If both AC are
     maneuvered, the direction of the vertical resolution advisory for each AC
     must be determined and stored in the RADS.>

    IF (both aircraft are to be maneuvered)
         THEN PERFORM two_AC_resolution_logic_vertical_resolution_advisories_
                                                                   selection;
              SET pointer to Resolution Advisory Data Set (RADS) with both
                   AC maneuvered;
    ELSEIF (first aircraft is maneuvered)
         THEN SET pointer to RADS with only first AC maneuvered;
    OTHERWISE SET pointer to RADS with only second AC maneuvered;

    LOOP;
         Get next advisory from list of all possible advisories;
    EXITIF (no more advisories);
         IF ((this advisory set contains a vertical component) AND
                   (both AC are maneuvered))
              THEN fill in vertical resolution advisory values;
              ELSE;
         IF (RADS contains a DESCEND resolution advisory that would
                   cause a descent below terrain altitude alert threshold)
              THEN SET flag indicating positive descend can not be given;
                   Correct vertical advisories in RADS for flag indicating
                        DESCEND can not be given, if so indicated;
              ELSE;
    ENDLOOP;

END potential_resolution_advisory_sets_selection;
```

```
--------------------------------------------------------------------------------
PROCESS potential_resolution_advisory_sets_selection;

    IF ((RSPND1 EQ $TRUE) AND (RSPND2 EQ $TRUE))
        THEN PERFORM two_AC_resolution_logic_vertical_resolution_advisories_
                                                                    selection;
             RADS = BACHRADS;
    ELSEIF (RSPND1 EQ $TRUE)
        THEN RADS = FACHRADS;
    OTHERWISE RADS = SACHRADS;

    LOOP;
        Get next advisory from list of all possible advisories;
    EXITIF (no more advisories);
        IF ((TRADS.CHDED_CHDED EQ $TRUE) AND (TRADS.VERT EQ $TRUE))
            THEN TRADS.V1 = VERTRA1;
                 TRADS.V2 = VERTRA2;
            ELSE;
        IF (either AC has a $DES AND
                 that AC's altitude is within ATERN of terrain)
            THEN TRADS.BELOW1000 = $TRUE;
                 Convert $DES to $DCL;
            ELSE;
    ENDLOOP;

END potential_resolution_advisory_sets_selection;
```

```
------------------------------------------------------------------------------------
PROCESS potential_resolution_advisory_sets_culling;

    <Determine if the negative sense of any of the advisory sets is sufficient.
     If any negative vertical advisories are sufficient, check for Vertical Speed
     Limit advisories being sufficient. Eliminate any advisories that
     do not meet the minimum criteria for selection. Keep a count of those
     RADS that do meet the minimum set of criteria. Keep a pointer to the
     advisories that meet the minimum criteria.>

    CLEAR pointer to selected resolution advisory set;
    CLEAR counter of potential resolution advisories with all absolute features set;

    LOOP;
        Get next RADS;     <Data structure with potential advisory>
    EXITIF (looked at every RADS);
        IF (this RA set is single dimension advisories)
            THEN PERFORM negative_resolution_advisory_determination;
            ELSE;
        IF (negative vertical advisories have been computed)
            THEN CALL VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION;
            ELSE;

        PERFORM absolute_features_evaluation_two_AC_resolution_definition;
        IF (all absolute features set)
            THEN increment potential resolution advisory counter;
                IF (pointer to selected resolution advisory set is null)
                    THEN SET pointer to selected resolution advisory set to
                            point to this resolution advisory set;
                    ELSE;
            ELSE;
    ENDLOOP;

END potential_resolution_advisory_sets_culling;
```

```
--------------------------------------------------------------------------------
PROCESS potential_resolution_advisory_sets_culling;

     RADSPTR = $NULL;
     NPRAABS = 0;

     LOOP;
          Get next RADS;     <Data Structure with potential advisory>
     EXITIF (looked at every RADS);
          IF (TRADS.SINGLE EQ $TRUE)
               THEN PERFORM negative_resolution_advisory_determination;
               ELSE;
          IF ((TRADS.NEGATIVE EQ $TRUE) AND (TRADS.VERT EQ $TRUE))
               THEN CALL VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION
                         IN (TRADS, ACID1, ACID2, PREC)
                         INOUT (TRADS.V1, TRADS.V2);
               ELSE;

          PERFORM absolute_features_evaluation_two_AC_resolution_definition;
          IF ((TRADS.FEATBITS(1) EQ $TRUE) AND (TRADS.FEATBITS(2) EQ $TRUE)
                    AND (TRADS.FEATBITS(3) EQ $TRUE))
               THEN NPRAABS = NPRAABS + 1;
                    IF (RADSPTR EQ $NULL)
                         THEN RADSPTR = TRADS;
                         ELSE;
               ELSE;
     ENDLOOP;

END potential_resolution_advisory_sets_culling;
```

```
--------------------------------------------------------------------------------
PROCESS multi_AC_resolution;

    <This process is called if no advisory sets meet the minimum criteria for
     selection.  The multi-AC logic determines additional sets of advisories
     from which a final set may be selected.

     Do not perform the multi_AC logic if RAER is called by the Conflict
     Resolution Data Task.  This is accomplished by setting the deliverable
     feature in this process for all resolution advisory sets.  The other
     two absolute features have already been set.>

    IF (RAER called from Conflict Resolution Data Task)
         THEN SET deliverable feature in all RADS;
              Count the number of resolution Advisory Data Sets with all
                   absolute features set;
         ELSE PERFORM resolution_advisory_compatibility_with_existing_conflicts;

    IF (any potential resolution advisories have all absolute features set)
         THEN;  <return to complete evaluation of the features.>
         ELSE IF (both AC maneuvered)
                   THEN PERFORM multi_AC_conflict_possible_resolution_advisories;
                        IF (any new potential resolution advisory sets exits)
                             THEN PERFORM multi_AC_resolution_logic_advisories_
                                                                   calculations;
                             ELSE;
                   ELSE;

END multi_AC_resolution;
```

```
--------------------------------------------------------------------------------
PROCESS multi_AC_resolution;

    IF (NRNCAP EQ $FALSE)
        THEN LOOP:
                Get next RADS;
            EXITIF (no more RADS);
                TRADS.FEATBITS(1) = $TRUE;
                NPRAABS = NPRAABS + 1;
            ENDLOOP;
        ELSE PERFORM resolution_advisory_compatibility_with_existing_conflicts;

    IF (NPRAABS GE 1)
        THEN: <return to complete evaluation of the features.>
        ELSE IF ((RSPND1 EQ $TRUE) AND (RSPND2 EQ $TRUE))
                THEN PERFORM multi_AC_conflict_possible_resolution_advisories;
                     PERFORM multi_AC_resolution_logic_advisories_calculations;
                ELSE;

END multi_AC_resolution;
```

```
--------------------------------------------------------------------------------
PROCESS final_resolution_advisory_selection;

     <Select the set of resolution advisories to be given to the conflict
      pair from among the set of advisories with all the absolute features set.
      This is done by first evaluating the relative features (except dominio) for
      all the potential resolution advisory sets.  If more than one advisory set is
      tied for selection, based on the features with higher weights than the domino
      feature, then evaluate the domino feature.  If the tie is still not broken,
      evaluate the tie breaker features.>

     PERFORM relative_features_evaluation;
     PERFORM highest_valued_potential_resolution_advisory_sets_count;

     IF (more than one potential resolution advisory set is tied for highest
              value of its features)
          THEN IF (RAER called from Master Resolution Task)
                    THEN PERFORM feature_domino_logic;
                         PERFORM highest_valued_potential_resolution_advisory_
                                                                    sets_count;
                    ELSE;
               IF (more than one potential resolution advisory set is tied for
                         the highest value)
                    THEN PERFORM tie_breaker_features_evaluation;
                    ELSE;
          ELSE;

     PERFORM PSEP_model_validation_values_saved;

END final_resolution_advisory_selection;
```

```
--------------------------------------------------------------------------------

PROCESS final_resolution_advisory_selection;

     PERFORM relative_features_evaluation;
     PERFORM highest_valued_potential_resolution_advisory_sets_count;

     IF (NUMPRA GT 1)
          THEN IF (NRNCAP EQ $TRUE)
                    THEN PERFORM feature_domino_logic;
                         PERFORM highest_value_potential_resolution_advisory_
                                                                sets_count;
                    ELSE;
               IF (NUMPRA GT 1)
                    THEN PERFORM tie_breaker_features_evaluation;
                    ELSE;
          ELSE;

     PERFORM PSEP_model_validation_values_saved;

END final_resolution_advisory_selection;
```

```
------------------------------------------------------------------------------------
PROCESS PSEP_model_validation_values_saved;

     <Save each AC's turn status and the relative vertical velocity. This data will
      be used on succeeding scans to determine if the conditions change in such a way
      that resolution advisories should be recalculated.>

     IF (RAER called from the Master Resolution Task)
          THEN CLEAR 'PSEP model validation performed' flag in pair record;
               Save turn status of each AC in pair record;
               Save vertical velocity difference in pair record;
               Save current time in pair record;
          ELSE;

END PSEP_model_validation_values_saved;
```

```
--------------------------------------------------------------------------------
PROCESS PSEP_model_validation_values_saved;

     IF (MRNCAP EQ $TRUE)
          THEN PREC.MVDONE = $FALSE;
               PREC.MVRAIT = SYSVAR.CTIME;
               PREC.MVVRZ = ACID2.ZD - ACID1.ZD;
               PREC.ac1.MVT = ACID1.TURN;
               PREC.ac2.MVT = ACID2.TURN;
          ELSE;

END PSEP_model_validation_values_saved;
```

```
--------------------------------------------------------------------------------
PROCESS absolute_features_evaluation_two_AC_resolution_definition;

    <Evaluate the three absolute features.  Evaluate the two-AC resolution logic
     definition of the maneuvered_unmaneuvered conflict feature.  If RAER is
     called by the Conflict Resolution Data Task, evaluate only the deliverable
     feature; the other two absolute features are set automatically.>

    PERFORM feature_deliverable;
    IF (RAER called from Master Resolution)
         THEN PERFORM feature_dimension_available;
              PERFORM feature_maneuvered_unmaneuvered_conflict_two_AC_definition;
         ELSE SET dimension available feature;
              SET maneuvered_unmaneuvered feature;

END absolute_features_evaluation_two_AC_resolution_definition;
```

```
--------------------------------------------------------------------------------

PROCESS absolute_features_evaluation_two_AC_resolution_definition;


    PERFORM feature_deliverable;
    IF (MRNCAP EQ $TRUE)
        THEN PERFORM feature_dimension_available;
             PERFORM feature_maneuvered_unmaneuvered_conflict_two_AC_definition;
        ELSE TRADS.FEATBITS(2) = $TRUE;
             TRADS.FEATBITS(3) = $TRUE;

END absolute_features_evaluation_two_AC_resolution_definition;
```

```
------------------------------------------------------------------------------------------
PROCESS absolute_features_evaluation_multi_AC_resolution_definition;

    <Evaluate the three absolute features for the additional potential resolution
     advisory sets determined by the multi-AC resolution logic.  Use the multi-AC
     resolution logic definition of the maneuverded_unmaneuvered conflict feature.>

    PERFORM feature_deliverable;
    PERFORM feature_dimension_available;
    PERFORM feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;

END absolute_features_evaluation_multi_AC_resolution_definition;
```

```
--------------------------------------------------------------------------------

PROCESS absolute_features_evaluation_multi_AC_resolution_definition;

     PERFORM feature_deliverable;
     PERFORM feature_dimension_available;
     PERFORM feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;

END absolute_features_evaluation_multi_AC_resolution_definition;
```

```
-------------------------------------------------------------------------------

PROCESS domino_coarse_detection_checks;

    <Perform coarse detection checks to determine if the aircraft may be in
     conflict.  If they may, then the detailed detection checks must be performed.
     The vertical dimension check is linear.  Therefore, the vertical coarse
     detection check can be performed from the first domino projected position.
     The horizontal dimension check is linear only if no positive horizontal
     resolution advisory is being examined.  All calculations are from the domino
     projected positions in the RAPP table and DOPP section of the state vector.>

    CLEAR domino resolution advisory flag;
    PERFORM domino_coarse_detection_vertical_dimension_checks;

    IF (vertical conflict possible)
        THEN CLEAR domino resolution advisory flag;
             CLEAR all domino DOT variables;
             LOOP:
                  Get next domino projected position;
                  PERFORM domino_coarse_detection_horizontal_dimension_checks;
             EXITIF (all projected positions processed OR domino conflict possible
                     OR horizontal resolution advisory is linear);
             ENDLOOP;

        ELSE:

END domino_coarse_detections_checks;
```

```
-------------------------------------------------------------------------------------

PROCESS domino_coarse_detection_checks;

     DCMDFLG = $FALSE;
     PERFORM domino_coarse_detection_vertical_dimension_checks;

     IF (DCMDFLG EQ $TRUE)
          THEN DCMDFLG = $FALSE;
               DDOT(*) = $UNTAU;
               LOOP;
                    Get next domino projected position;
                    PERFORM domino_coarse_detection_horizontal_dimension_checks;
               EXITIF ((checked all domino projected positions) OR
                         (possible domino conflict has already been detected) OR
                         (TRADS.HORIZ EQ $FALSE) OR (TRADS.NEGATIVE EQ $TRUE));
               ENDLOOP;

          ELSE;

END domino_coarse_detections_checks;
```

```
PROCESS domino_coarse_detection_horizontal_dimension_checks;

    <Perform coarse detection checks in the horizontal dimension.  A conflict is
     possible if the horizontal tau threshold or if the immediate range threshold
     will be violated within the domino projection interval.>

    Compute DOT;
    Compute horizontal tau;

    IF (zero LT horizontal tau LT horizontal tau threshold)
         THEN SET flag for possible conflict;
         ELSE IF (AC are diverging horizontally)
                   THEN IF (AC currently within immediate horizontal
                                 range threshold)
                             THEN SET flag for possible conflict;
                             ELSE;

                   ELSE compute time to separation of immediate
                             range threshold;
                        IF (time to threshold violation is within the horizontal
                                 tau threshold)
                             THEN SET flag for possible conflict;
                             ELSE;

END domino_coarse_detection_horizontal_dimension_checks;
```

```
--------------------------------------------------------------------------------

PROCESS domino_coarse_detection_horizontal_dimension_checks;

    FLT TWO                     <local constant: 2.0>
    FLT (T1, T2, DT1);

    DDOT = ((ACID.X - PDC_LIST.INTRAC.XPRJ(n)) *
                 (ACID.XD - PDC_LIST.INTRAC.XDPRJ(n))) +
            ((ACID.Y - PDC_LIST.INTRAC.YPRJ(n)) *
                 (ACID.YD - PDC_LIST.INTRAC.YDPRJ(n)));

    DDSQ = DETPARM.BDET + (DETPARM.ADET * (ACID.VSQ + PDC_LIST.INTRAC.VSQ));
    DRANGE2 = (RAPP.X(subject maneuver,1) - PDC_LIST.INTRAC.XPRJ(1))**2 +
              (RAPP.Y(subject maneuver,1) - PDC_LIST.INTRAC.YPRJ(1))**2;

    DTH = -(DRANGE2 - DDSQ) / DDOT;

    IF ((0 LT DTH) AND (DTH LT DTCMDH))
        THEN DCMDFLG = STRUE;
        ELSE IF (DTH LT 0)
                THEN IF (DRANGE2 LE DRCMD2)
                        THEN DCMDFLG = STRUE;
                        ELSE;

                ELSE T1 = -(DRANGE2 - DDSQ) - DRCMD2;
                     T2 = T1 + (TWO * DRCMD2);
                     DT1 = MIN(T1/DDOT,T2/DDOT);
                     IF (DT1 LT DTCMDH)
                         THEN DCMDFLG = STRUE;
                         ELSE;

END domino_coarse_detection_horizontal_dimension_checks;
```

```
--------------------------------------------------------------------------------

PROCESS domino_coarse_detection_vertical_dimension_checks;

    <Perform coarse detection checks in the vertical dimension.  A conflict is
     possible if the vertical tau threshold or the immediate altitude threshold
     will be violated within the domino projection interval.>

    Calculate vertical tau;

    IF (vertical tau LT vertical tau threshold within next 3 scans)
        THEN SET flag for possible conflict;
        ELSE IF (AC are diverging vertically)
                THEN IF (AC within immediate altitude separation threshold)
                        THEN SET flag for possible conflict;
                        ELSE;
                ELSE compute time to vertical separation of immediate
                        altitude separation threshold;
                    IF (time to threshold violation is within 3 scans)
                        THEN SET flag for possible conflict;
                        ELSE;

END domino_coarse_detection_vertical_dimension_checks;
```

```
--------------------------------------------------------------------------------

PROCESS domino_coarse_detection_vertical_dimension_checks;

     FLT (T1, T2, DT1);

     DRZ = RAPP.Z(subject maneuver,1) - PDC_LIST.INTRAC.ZPRJ(1);
     DVRZ = RAPP.ZD(subject maneuver,1) - PDC_LIST.INTRAC.ZD;
     DTV = -DRZ / DVRZ;
     DALT = ABS(RAPP.Z(subject maneuver,1) - PDC_LIST.INTRAC.ZPRJ(1));

     IF (DTV LT 0)
          THEN IF (DALT LE DAF)
                    THEN DCMDFLG = $TRUE;
                    ELSE;
          ELSE T1 = -DALT - DAF;
               T2 = -DALT + DAF;
               DT1 = MIN(T1/DVRZ,T2/DVRZ);
               IF (DT1 LT (DOMCRSE * SYSTEM.SCANT));
                    THEN DCMDFLG = $TRUE;
                    ELSE;

END domino_coarse_detection_vertical_dimension_checks;
```

```
---------------------------------------------------------------------------------
PROCESS domino_coarse_screen;

     <Calculate the Domino Coarse Screen search limits for the subject aircraft and
      compile a list of those aircraft on the X-list and/or EX-list within the
      search limits.  The search limits consist of the area covered by the subject
      AC during the domino projection interval plus a buffer to allow for the maximum
      area covered by a domino object AC during the domino projection interval.>

     PERFORM potential_resolution_advisory_status_variable_determination;
     IF (subject AC on EX-list)
          THEN SET maximum detection threshold parameter to the appropriate value;
               PERFORM domino_search_area_subject_AC_calculations;
               PERFORM EX_list_object_AC_domino_buffer_area_calculations;
               PERFORM EX_list_domino_search_limits_calculations;
               PERFORM EX_list_domino_search;

               IF (subject AC within altitude range of X-list)
                    THEN PERFORM X_list_object_AC_domino_buffer_area_calculations;
                         PERFORM X_list_domino_search_limits_calculations;
                         CALL X_LIST_SIGNPOST_ENTRY_CALCULATION
                                 IN (X position of subject AC)
                                 OUT (signpost entry);
                         Use signpost entry as subject AC;
                         PERFORM X_list_domino_search;
                    ELSE;

          ELSE IF (subject AC controlled)
                    THEN SET maximum threshold parameter to appropriate IFR value;
                    ELSE SET maximum threshold parameter to appropriate VFR value;
               PERFORM domino_search_area_subject_AC_calculations;
               PERFORM X_list_object_AC_domino_buffer_area_calculations;
               PERFORM X_list_domino_search_limits_calculations;
               PERFORM X_list_domino_search;

END domino_coarse_screen;
```

```
---------------------------------------------------------------------------------

PROCESS domino_coarse_screen;

     PERFORM potential_resolution_advisory_status_variable_determination;
     IF (ACID.EXFLG EQ $TRUE)
          THEN TLD = MAXTLI;
               PERFORM domino_search_area_subject_AC_calculations;
               PERFORM EX_list_object_AC_domino_buffer_area_calculations;
               PERFORM EX_list_domino_search_limits_calculations;
               PERFORM EX_list_domino_search;
               IF (ACID.Z LT (CSCREEN.ALO + MAXAF))
                    THEN PERFORM X_list_object_AC_domino_buffer_area_calculations;
                         PERFORM X_list_domino_search_limits_calculations;
                         CALL X_LIST_SIGNPOST_ENTRY_CALCULATION
                                 IN (ACID.X)
                                 OUT (SIGNPOST);
                         Use signpost entry as subject AC;
                         PERFORM X_list_domino_search;
                    ELSE;

          ELSE IF (ACID.CUNC EQ $TRUE)
                    THEN TLD = MAXTLI;
                    ELSE TLD = MAXTLV;
               PERFORM domino_search_area_subject_AC_calculations;
               PERFORM X_list_object_AC_domino_buffer_area_calculations;
               PERFORM X_list_domino_search_limits_calculations;
               PERFORM X_list_domino_search;

END domino_coarse_screen;
```

```
PROCESS domino_coarse_screen_altitude_conflict_test;

     <This process checks for AC on the X and/or EX-list within the Domino Coarse
      Screen altitude limits.  If the object AC has altitude data, then the AC must
      either be within the Domino Coarse Screen altitude limits or, if it has a
      vertical velocity greater than the assumed vertical velocity, the two AC must
      be co-altitude within the detection tau threshold.  If the object AC does not
      have altitude data, it is considered within the altitude limits.>

     IF (next AC has altitude data)
          THEN IF (next AC Z position within Z search limits)
                    THEN PERFORM potential_domino_conflict_list_entry_addition;
                    ELSE IF (next AC Z velocity GT assumed maximum Z velocity)
                              THEN IF (next AC and subject AC will be co-altitude
                                                  within maximum detection threshold)
                                        THEN PERFORM potential_domino_conflict_list_
                                                                      entry_addition;
                                        ELSE;
                              ELSE;
          ELSE IF (non-mode C domino processing flag is set)
                    THEN PERFORM potential_domino_conflict_list_entry_addition;
                    ELSE;

END domino_coarse_screen_altitude_conflict_test;
```

```
--------------------------------------------------------------------------------

PROCESS domino_coarse_screen_altitude_conflict_test;

     IF (ACID.MCFLG EQ $TRUE)
          THEN IF ((ZL LT ACID.Z) AND (ACID.Z LT ZU))
                    THEN PERFORM potential_domino_conflict_list_entry_addition;
                    ELSE IF (ACID.ZD GT CSCREEN.ZFAST)
                              THEN IF (ACID.Z - NXTAC.Z)/(NXTAC.ZD - ACID.ZD))
                                             LT TLD)
                                        THEN PERFORM potential_domino_conflict_
                                                                 list_entry_addition;
                                        ELSE;
                              ELSE;
          ELSE IF (SYSTEM.DOMNONC EQ $TRUE)
                    THEN PERFORM potential_domino_conflict_list_entry_addition;
                    ELSE;

END domino_coarse_screen_altitude_conflict_test;
```

```
PROCESS domino_conflict_detection;

     <Determine a list of potential domino conflict AC for the subject AC.
      Using the subject AC's modeled response path to each potential resolution
      advisory and the object AC's projected path, perform resolution advisory
      detection checks between the subject aircraft and each AC on the Potential
      Domino Conflict List, until a conflict requiring resolution advisories is
      detected or every AC on the Potential Domino Conflict List has been processed.>

     IF (potential domino conflict list pointer in pair record for subject AC
               is null)
          THEN PERFORM potential_domino_conflict_list_creation;
          ELSE;     <Potential Domino Conflict List already determined>

     IF (this advisory has not been checked for causing a domino conflict)
          THEN LOOP;
                    Get next AC on the potential domino conflict list;
               EXITIF (no more AC OR potential resolution advisory status variable
                         indicates this potential resolution advisory has already
                         been checked and it causes a domino conflict);
                    PERFORM domino_resolution_advisory_detection_filter;
                    IF (conflict requiring a resolution advisory is detected)
                         THEN SET potential resolution advisory domino status
                                   variable to indicate domino conflict detected;
                         ELSE;
               ENDLOOP;
               IF (potential resolution advisory domino status variable does not
                         indicate a domino conflict has been detected)
                    THEN SET potential resolution advisory domino status variable to
                              indicate no domino conflict detected;
                    ELSE;
          ELSE;

END domino_conflict_detection;
```

AD-A104 148 MITRE CORP MCLEAN VA METREK DIV F/G 17/7

AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)

JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370

UNCLASSIFIED MTR-81W120-2 FAA-RD-81-45-2 NL

3 OF 7

AD
A104148

```
--------------------------------------------------------------------------------
PROCESS domino_conflict_detection;

     <Using the subject AC's modeled response path to each potential resolution
      advisory, perform RA detection checks between the subject AC
      and each AC on the Potential Domino Conflict List, until a conflict requiring
      RAS is detected or every AC on the Potential Domino Conflict
      List has been processed.>

     IF (PREC.INTR EQ $NULL)
          THEN PERFORM potential_domino_conflict_list_creation;
          ELSE;

     IF (potential resolution advisory status variable EQ $DOMNP)
          THEN LOOP:
                    Get next AC on the potential domino conflict list;
               EXITIF ((no more AC) OR (potential RA status variable EQ $DOMCC))
                    PERFORM domino_resolution_advisory_detection_filter;
                    IF (DCMDFLG EQ $TRUE)
                         THEN potential RA domino status variable = $DOMCC;
                         ELSE;
               ENDLOOP;

               IF (potential RA domino status variable NE $DOMCC)
                    THEN SET potential RA domino status variable = $DOMCNC;
                    ELSE;
          ELSE;

END domino_conflict_detection;
```

```
--------------------------------------------------------------------------------
PROCESS domino_detection_thresholds:

    <Determine the detection thresholds to be used for the subject AC and the object
     AC from the Potential Domino Conflict List.  The determining factors are the
     controlled/uncontrolled, equipped/unequipped status of each AC, the number of
     AC in the conflict cluster, and the encounter area type.  The area type of the
     intruder AC must be determined if it is not already known.>

    IF (encounter multiplicity is not known)
        THEN PERFORM domino_encounter_AC_multiplicity_determination;
        ELSE;

    IF (encounter area type is not known for subject AC, object AC pair)
        THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
        ELSE;

    Calculate the absolute value of the relative vertical velocity;
    CALL AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION;

    CALL DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;

END domino_detection_thresholds;
```

```
--------------------------------------------------------------------------------
PROCESS domino_detection_thresholds;

     <Determine the detection thresholds to be used for the subject AC and the object
      AC from the Potential Domino Conflict List.  The determining factors are the
      controlled/uncontrolled and equipped/unequipped status of each AC.  Also, the
      area type of the intruder AC must be determined if it is not already known.>

     IF (PDC_LIST.MULT EQ 0)
          THEN PERFORM domino_encounter_AC_multiplicity_determination;
          ELSE:

     IF (PDC_LIST.ENAT EQ SUNAT)
          THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION
                         IN (ACID1, ACID2)
                         OUT (DENAT, INFAZ2);
               PDC_LIST.ENAT = DENAT;
          ELSE:

     AVRZ = ABS(DVRZ);
     CALL AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION
           IN (ACID1, ACID2)
           OUT (PRCONT, PREQ);

     CALL DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
           IN (AVRZ, PDC_LIST.ENAT, PDC_LIST.MULT, PREQ, PRCONT, DDOT)
           OUT (STRUCTURE DRAVBL);

END domino_detection_thresholds;
```

```
--------------------------------------------------------------------------------
PROCESS domino_encounter_AC_multiplicity_determination;

     <This process determines the total number of AC involved in the conflict
      cluster including the domino object AC.>

     IF (domino object AC is not in any conflict table)
          THEN multiplicity is number of AC in subject AC conflict table plus
                    one for domino object AC;
     ELSEIF (domino object AC is not in the same conflict table as
               the subject AC)
          THEN multiplicity is sum of number of AC in each conflict table;
     OTHERWISE multiplicity is number of AC in subject AC conflict table;

END domino_encounter_AC_multiplicity_determination;
```

```
-----------------------------------------------------------------------------------

PROCESS domino_encounter_AC_multiplicity_determination;

     <This process determines the total number of AC, including the domino object AC
      involved in the conflict cluster with which the subject pair is associated.>

     IF (PDC_LIST.INTRAC.CTPTR EQ $NULL)
          THEN PDC_LIST.MULT = ACID.CTPTR.NAC + 1;
     ELSEIF (PDC_LIST.INTRAC NE ACID.CTPTR)
          THEN PDC_LIST.MULT = ACID.CTPTR.NAC + PDC_LIST.INTRAC.CTPTR.NAC;
     OTHERWISE PDC_LIST.MULT = ACID.CTPTR.NAC;

END domino_encounter_AC_multiplicity_determination;
```

```
--------------------------------------------------------------------------------

PROCESS domino_features_weight_addition;

     <Add the weight of any domino features set to that RADS total value.>

     LOOP;
          Get next RADS;
     EXITIF (no more RADS);
          IF ('neither AC causes a domino conflict' feature is set)
               THEN add this feature's weight to this RADS total value;
          ELSEIF ('one AC causes a domino conflict' feature is set)
               THEN add this feature's weight to this RADS total value;
          OTHERWISE;
     ENDLOOP;

END add_domino_features_weight;
```

```
--------------------------------------------------------------------------------
PROCESS domino_features_weight_addition;

    LOOP:
        Get next RADS;
    EXITIF (no more RADS);
        IF (TRADS.FEATBITS(7) EQ $TRUE)
            THEN TRADS.VALUE = TRADS.VALUE + NDONWGT;
        ELSEIF (TRADS.FEATBITS(8) EQ $TRUE)
            THEN TRADS.VALUE = TRADS.VALUE + DOM1WGT;
        OTHERWISE;
    ENDLOOP;

END domino_features_weight_addition;
```

```
PROCESS domino_resolution_advisory_detection_filter;

     <This routine checks for a domino conflict between the subject AC and object AC
      caused by a potential resolution advisory.  The resolution advisory detection
      thresholds must first be determined.  Then the coarse detection checks are
      performed.  If a domino conflict is possible, then the detection logic is
      performed.  The non-mode C detection logic is performed if the object AC does
      not have mode C data.>

     CLEAR resolution advisory detection flag;
     PERFORM domino_detection_thresholds;

     PERFORM domino_coarse_detection_checks;
     IF (conflict still possible)
          THEN LOOP;
                    Get index of next domino projected position in RAPP Table and
                       DOPP Table;
               EXITIF (all projected positions have already been tested OR
                         flag indicating necessity for resolution advisory
                         is set);

                    IF (mode C data available for object AC)
                         THEN CALL DOMINO_RESOLUTION_TAU_AND_PROXIMITY_
                                                            COMPARISONS;
                         ELSE PERFORM non_mode_C_resolution_tau_and_
                                                  proximity_comparisons;
               ENDLOOP;
          ELSE;

END domino_resolution_advisory_detection_filter;
```

```
--------------------------------------------------------------------------------
PROCESS domino_resolution_advisory_detection_filter:

    <This is the detection logic for determining the setting of the potential
     RA status variables from the value of the DCMDFLG.  Only one flag, DCMDFLG,
     is processed.  If both AC are controlled, the values of DTCMDH, V are actually
     TIFRH, V.  If both AC are uncontrolled, then the DCMDFLG is the only flag that
     should be processed.  And if one AC is controlled and one AC is uncontrolled,
     it is sufficient to process only the more sensitive (larger) thresholds.>

    DCMDFLG = $FALSE;
    PERFORM domino_detection_thresholds;

    PERFORM domino_coarse_detection_checks;
    IF (DCMDFLG EQ $TRUE)
        THEN DCMDFLG = $FALSE;
            LOOP:
                Get index of next domino projected position in RAPP Table &
                    DOPP Table;
            EXITIF (all projected positions have already been tested OR
                    DCMDFLG EQ $TRUE);
                    IF (ACID.MCFLG EQ $TRUE)
                        THEN CALL DOMINO_RESOLUTION_TAU_AND_PROXIMITY_
                                                             COMPARISONS
                                   IN (STRUCTURE DRABVL)
                                   OUT (DCMDFLG);
                        ELSE PERFORM non_mode_C_resolution_tau_and_
                                                  proximity_comparisons;
            ENDLOOP:
        ELSE:

END domino_resolution_advisory_detection_filter;
```

```
------------------------------------------------------------------------------------
PROCESS domino_search_area_horizontal_dimension_calculations;

    <Determine the extent of the subject AC during the domino interval in the
     horizontal dimension.  For the 'continue straight' path, the projection is
     linear and may be made from the first domino projected position.  For the
     turning paths, a projection must be made from each of the domino projected
     positions during the interval.>

    CLEAR all horizontal projection values;
    IF (negative horizontal advisory is a potential resolution advisory OR
              any vertical only advisory is a potential resolution advisory)
         THEN project subject AC ahead on the continue straight path from the first
                   RAPP Table entry for 3 * scan time + max horizontal tau
                   detection threshold;
         ELSE;

    IF (turn left advisory is a potential resolution advisory)
         THEN project subject AC ahead on the turn left path from all four entries
                   in the RAPP Table for max horiz tau detection threshold;
         ELSE;

    IF (turn right is a potential resolution advisory)
         THEN project subject AC ahead on the turn right path from all four entries
                   in the RAPP Table for max horiz tau detection threshold;
         ELSE;

END domino_search_area_horizontal_dimension_calculations;
```

```
--------------------------------------------------------------------------------
PROCESS domino_search_area_horizontal_dimension_calculations;

    XPR(*) = $UNPOS;
    YPR(*) = $UNPOS;

    IF ((NLFTNRGT EQ $DOMNP) OR (CLMB EQ $DOMNP) OR (DSC EQ $DOMNP) OR
            (NCLMB EQ $DOMNP) OR (NDSC EQ $DOMNP) OR (VSL EQ $DOMNP))
        THEN XPR(1) = RAPP.X(1,1) + RAPP.XD(1,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
             YPR(1) = RAPP.Y(1,1) + RAPP.YD(1,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

    IF ((LFT EQ $DOMNP) OR (LFTCLMB EQ $DOMNP) OR (LFTDSC EQ $DOMNP))
        THEN XPR(2) = RAPP.X(2,1) + RAPP.XD(2,1) * TLD;
             YPR(2) = RAPP.Y(2,1) + RAPP.YD(2,1) * TLD;
             XPR(3) = RAPP.X(2,2) + RAPP.XD(2,2) * TLD;
             YPR(3) = RAPP.Y(2,2) + RAPP.YD(2,2) * TLD;
             XPR(4) = RAPP.X(2,3) + RAPP.XD(2,3) * TLD;
             YPR(4) = RAPP.Y(2,3) + RAPP.YD(2,3) * TLD;
             XPR(5) = RAPP.X(2,4) + RAPP.XD(2,4) * TLD;
             YPR(5) = RAPP.Y(2,4) + RAPP.YD(2,4) * TLD;
        ELSE;

    IF ((RGT EQ $DOMNP) OR (RGTCLMB EQ $DOMNP) OR (RGTDSC EQ $DOMNP))
        THEN XPR(6) = RAPP.X(3,1) + RAPP.XD(3,1) * TLD;
             YPR(6) = RAPP.Y(3,1) + RAPP.YD(3,1) * TLD;
             XPR(7) = RAPP.X(3,2) + RAPP.XD(3,2) * TLD;
             YPR(7) = RAPP.Y(3,2) + RAPP.YD(3,2) * TLD;
             XPR(8) = RAPP.X(3,3) + RAPP.XD(3,3) * TLD;
             YPR(8) = RAPP.Y(3,3) + RAPP.YD(3,3) * TLD;
             XPR(9) = RAPP.X(3,4) + RAPP.XD(3,4) * TLD;
             YPR(9) = RAPP.Y(3,4) + RAPP.YD(3,4) * TLD;
        ELSE;

END domino_search_area_horizontal_dimension_calculations;
```

```
------------------------------------------------------------------------------

PROCESS domino_search_area_subject_AC_calculations;

     <Calculate the extent of the subject AC on the X/FX-list horizontally and
      vertically based on response to the potential resolution advisories
      and the maximum resolution advisory detection threaholds.>

     CLEAR projected X, Y and Z values;
     PERFORM domino_search_area_horizontal_dimension_calculations;

     <Check to see if negative vertical advisories must be modeled for later
      domino detection processing.>

     IF (subject AC has a vertical speed limit advisory or a negative vertical
               resolution advisory that has not yet been modeled)
          THEN CALL NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING;
          ELSE;
     PERFORM domino_search_area_vertical_dimension_calculations;

END domino_search_area_subject_AC_calculations;
```

```
--------------------------------------------------------------------------------
PROCESS domino_search_area_subject_AC_calculations;

    <Calculate the domino area on the X/EX-list around the subject AC based on
     response to the potential RAS (in the RAPP Table) and the
     maximum RA detection thresholds.

    SET all XPR, YPR, ZPR to $UNPOS;
    PERFORM domino_search_area_horizontal_dimension_calculations;

    <Determine if negative vertical advisories must be modeled for later
     domino detection processing.>

    IF (((VSL EQ $DOMNP) OR (NCLMB EQ $DOMNP) OR (NDSC EQ $DOMNP))
              AND ((RSPND1 EQ $FALSE) OR (RSPND2 EQ $FALSE)))
         THEN CALL NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING
                       IN (ACID, VERTRA)
                       OUT (RAPP);
         ELSE;
    PERFORM domino_search_area_vertical_dimension_calculations;


    XU = MAX(XPR(*));
    XL = MIN(XPR(*));
    YU = MAX(YPR(*));
    YL = MIN(YPR(*));
    ZU = MAX(ZPR(*));
    ZL = MIN(ZPR(*));

END domino_search_area_subject_AC_calculations;
```

```
-------------------------------------------------------------------------------
PROCESS domino_search_area_vertical_dimension_calculations;

     <Determine the extent of the subject AC during the domino interval in the
      vertical dimension.  Since all paths are linear, the projections may be made
      from the first domino projected position for each path.>

     IF (any horizontal-only advisories are potential resolution advisories)
          THEN project subject AC ahead on current vertical path from the
                    first RAPP Table entry for 3 * scan time + max vert tau
                    detection threshold;
          ELSE;

     IF (climb is a potential resolution advisory)
          THEN project first entry in RAPP Table ahead 3 * scan time + max
                    vertical tau detection threshold;
          ELSE;

     IF (descend is a potential resolution advisory)
          THEN project first entry in RAPP Table ahead 3 * scan time +
                    max vertical tau detection threshold;
          ELSE;

     IF (a don't descend or limit descend is a potential resolution advisory)
          THEN project first entry in RAPP table ahead 3 * scan time +
                    max vertical tau detection threshold;
          ELSE;

     IF (a don't climb or limit climb is a potential resolution advisory)
          THEN project first entry in RAPP table ahead 3 * scan time +
                    max vertical tau detection threshold;
          ELSE;

END domino_search_area_vertical_dimension_calculations;
```

```
-------------------------------------------------------------------------------------------
PROCESS domino_search_area_vertical_dimension_calculations;

    IF ((LFT EQ $DOMNP) OR (RGT  EQ $DOMNP) OR (NLFTNRGT EQ $DOMNP))
        THEN ZPR(1) = RAPP.Z(1,1) + RAPP.ZD(1,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

    IF ((CLMB EQ $DOMNP) OR (LFTCLMB EQ $DOMNP) OR (RGTCLMB EQ $DOMNP))
        THEN ZPR(2) = RAPP.Z(2,1) + RAPP.ZD(2,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

    IF ((DSC EQ $DOMNP) OR (LFTDSC EQ $DOMNP) OR (RGTDSC EQ $DOMNP))
        THEN ZPR(3) = RAPP.Z(3,1) + RAPP.ZD(3,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

    IF (NDSC EQ $DOMNP)
        THEN ZPR(4) = RAPP.Z(4,1) + RAPP.ZD(4,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

    IF (NCLMB EQ $DOMNP)
        THEN ZPR(5) = RAPP.Z(5,1) + RAPP.ZD(5,1) * (DOMSRCH * SYSTEM.SCANT + TLD);
        ELSE;

END domino_search_area_vertical_dimension_calculations;
```

```
------------------------------------------------------------------------------
PROCESS EX_list_backward_domino_search;

     <Search backwards (decreasing X values) on the EX-list until the lower domino
      search limit is reached or there are no more AC.  Do not include state vectors
      that are signposts or AC that are currently in conflict with the subject AC.
      Also, don't include AC in a final approach zone if the subject AC is also
      in a final approach zone.>

     LOOP;
          Get next AC in direction of decreasing X on EX-list;
     EXITIF ((no more AC) OR (X position of next AC LT lower X limit));
          IF ((next AC not in a conflict pair with the subject AC) AND
                    (next state vector is not a signpost) AND
                    (both AC are not in final approach zones))
               THEN IF (next AC Y position within Y search limits)
                         THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                         ELSE;
               ELSE;
     ENDLOOP;

END EX_list_backward_domino_search;
```

```
--------------------------------------------------------------------------------

PROCESS EX_list_backward_domino_search;

    LOOP:
        Get next AC in direction of decreasing X on EX-list;
    EXITIF ((no more AC) OR (X position of next AC LT lower X limit));
        IF ((next AC not in a conflict pair with the subject AC) AND
                (NXTAC.SPIDFG EQ $FALSE) AND
                (both AC are not in a final approach zone))
            THEN IF ((YL LT NXTAC.Y) AND (NXTAC.Y LT YU))
                    THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                    ELSE;
            ELSE;
    ENDLOOP;

END EX_list_backward_domino_search;
```

```
--------------------------------------------------------------------------------

PROCESS EX_list_domino_search;

     <This procedure performs the search of the EX-list around the
      subject AC within the domino coarse screen search limits.>

     PERFORM EX_list_forward_domino_search;

     PERFORM EX_list_backward_domino_search;

END EX_list_domino_search;
```

```
--------------------------------------------------------------------------------
PROCESS EX_list_domino_search;

     PERFORM EX_list_forward_domino_search;

     PERFORM EX_list_backward_domino_search;

END EX_list_domino_search;
```

```
--------------------------------------------------------------------------------

PROCESS EX_list_domino_search_limits_calculations;

     <Calculate the EX-list domino search limits by adding the EX-list object AC
      domino buffer area to the subject AC domino area.>

     Add maximum horizontal range for object AC to upper X & Y values of subject AC
          domino area;

     Subtract maximum horizontal range for object AC from lower X & Y values of
          subject AC domino area;

     Add maximum vertical range for object AC to upper Z value of subject AC domino
          area;

     Subtract maximum vertical range from lower Z value of subject AC domino area;

END EX_list_domino_search_limits_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS EX_list_domino_search_limits_calculations;

    XU = XMAX + RMAX;
    YU = YMAX + RMAX;
    ZU = ZMAX + ZMX;

    XL = XMIN - RMAX;
    YL = YMIN - RMAX;
    ZL = ZMIN - ZMX;

END EX_list_domino_search_limits_calculations;
```

```
--------------------------------------------------------------------------------
PROCESS EX_list_forward_domino_search;

     <Search forward (increasing X values) on the EX-list until the upper domino
      search limit is reached or there are no more AC.  Do not include state vectors
      that are signposts or AC that are currently in conflict with the subject AC.
      Also, don't include AC in a final approach zone if the subject AC is also
      in a final approach zone.>

     LOOP:
         Get next AC in direction of increasing X on EX-list;
     EXITIF (no more AC OR X position of next AC GT upper X limit);
         IF ((next AC not in a conflict pair with the subject AC) AND
                  (next state vector is not a signpost) AND
                  (both AC are not in final approach zones))
              THEN IF (next AC Y position within Y search limits)
                        THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                        ELSE;
              ELSE;
     ENDLOOP;

END EX_list_forward_domino_search;
```

```
------------------------------------------------------------------------------------
PROCESS EX_list_forward_domino_search;

     LOOP;
          Get next AC in direction of increasing X on EX-list;
     EXITIF (no more AC OR X position of next AC GT upper X limit);
          IF ((next AC not in a conflict pair with the subject AC) AND
                    (NXTAC.SPIDFG EQ $FALSE) AND
                    (both AC are not in a final approach zone))
               THEN IF ((YL LT NXTAC.Y) AND (NXTAC.Y LT YU))
                         THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                         ELSE;
               ELSE;
     ENDLOOP;

END EX_list_forward_domino_search;
```

```
--------------------------------------------------------------------------------
PROCESS EX_list_object_AC_domino_buffer_area_calculations;

     <Calculate the maximum distance that an AC on the EX-list can travel
      during the domino projection interval.  This distance is based on
      the maximum speed of an AC on the EX-list, an assumed vertical
      velocity and the maximum detection threshold values.  The max immediate
      range threshold must be added to the horizontal range.>

     Calculate the maximum horizontal range as: max EX-List speed limit *
          (modeling delay period + 4 * scan time + max detection threshold) +
          max immediate range threshold;

     Calculate maximum vertical range as: max vertical speed limit *
               (modeling delay period + 4 * scan time + max detection threshold);

END EX_list_object_AC_domino_buffer_area_calculations;
```

```
PROCESS EX_list_object_AC_domino_buffer_area_calculations;

     RMAX = EXVEL * (DELAY + DOMSCANS * SYSTEM.SCANT + TLD) + PDVBL.RCONTH(3);

     ZMX = CSCREEN.ZFAST * (DELAY + DOMSCANS * SYSTEM.SCANT + TLD);

END EX_list_object_AC_domino_buffer_area_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS feature_aircraft_far_from_radar;

     <If either AC is 'far' from the radar, azimuth data becomes less reliable.
      Therefore, attempt to resolve the conflict using vertical-only advisories.>

     IF (either AC is far from the radar)
          THEN LOOP:
                    Get next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (this is a vertical dimension only resolution advisory set)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                         ELSE:
               ENDLOOP:
          ELSE:

END feature_aircraft_far_from_radar;
```

```
--------------------------------------------------------------------------------
PROCESS feature_aircraft_far_from_radar:

    IF (((ACID1.X**2 + ACID1.Y**2) GT RDISTR) OR
              (ACID**2 + ACID2.Y**2) GT RDISTR))
        THEN LOOP:
                 Get the next resolution advisory data set;
             EXITIF (all resolution advisory data sets processed);
                 IF (TRADS.HORIZ EQ $FALSE)
                     THEN TRADS.FEATBITS(9) = $TRUE;
                          TRADS.VALUE = TRADS.VALUE + FARRAWGT;
                     ELSE;
             ENDLOOP;
        ELSE;

END feature_aircraft_far_from_radar;
```

```
--------------------------------------------------------------------------------------
PROCESS feature_aircraft_on_final_approach;

    <If either maneuvered AC is in a final approach zone and is slow, attempt to
     resolve the conflict using horizontal-only advisories.>

    IF (either AC is in a final approach zone AND that AC is maneuvered
            AND that AC has a velocity that is not fast)
        THEN LOOP;
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF (this is a horizontal dimension only set)
                    THEN SET this feature;
                        Add this feature's weight to this RADS total value;
                    ELSE;
            ENDLOOP;
        ELSE;

END feature_aircraft_on_final_approach;
```

```
--------------------------------------------------------------------------------
PROCESS feature_aircraft_on_final_approach;

    IF (((ACID1.FAZ NE $FAZO) AND (RSPND1 EQ $TRUE) AND (ACID1.VSQ LT VFASTSQ))
            OR ((ACID2.FAZ NE $FAZO) AND (RSPND2 EQ $TRUE) AND
            (ACID2.VSQ LT VFASTSQ)))
        THEN LOOP:
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF (TRADS.VERT EQ $FALSE)
                    THEN TRADS.FEATBITS(16) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + FAZWGT;
                    ELSE:
            ENDLOOP:
        ELSE:

END feature_aircraft_on_final_approach;
```

```
-----------------------------------------------------------------------------------
PROCESS feature_big_horizontal_miss_distance;

     <If an advisory set with a horizontal component is projected to provide a
      large horizontal separation, attempt to use that advisory set to resolve the
      conflict.  The separation threshold used is larger if either of the AC are
      sensed to be turning by the tracker.>

     IF (neither AC is turning)
          THEN use default separation threshold;
          ELSE use larger separation threshold;

     LOOP;
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (there is a horizontal component to this set)
               THEN IF (modeled horizontal separation GT threshold)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_big_horizontal_miss_distance;
```

```
--------------------------------------------------------------------------------

PROCESS feature_big_horizontal_miss_distance;

    IF ((ACID1.TURN NE $STRNGLFT) AND (ACID1.TURN NE $STRNGRGT) AND
            (ACID2.TURN NE $STRNGLFT) AND (ACID2.TURN NE $STRNGRGT))
        THEN MDHM = MDHSQ;
        ELSE MDHM = MDHMSQ;

    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (TRADS.HORIZ EQ $TRUE)
            THEN IF (HMD2(TRADS.INDEX1,TRADS.INDEX2) GT MDHM)
                    THEN TRADS.FEATBITS(21) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + BIGHWGT;
                    ELSE;
            ELSE;
    ENDLOOP;

END feature_big_horizontal_miss_distance;
```

```
------------------------------------------------------------------------------------------
PROCESS feature_big_vertical_miss_distance;

    <If an advisory set with a vertical component is projected to provide a large
     vertical separation, attempt to use that advisory set to resolve the
     conflict.>

    LOOP;
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (there is a vertical component to this set)
            THEN IF (modeled vertical separation GT threshold)
                    THEN SET this feature;
                        Add this feature's weight to this RADS total value;
                    ELSE;
            ELSE;
    ENDLOOP;

END feature_big_vertical_miss_distance;
```

```
---------------------------------------------------------------------------------
PROCESS feature_big_vertical_miss_distance;

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (TRADS.VERT EQ $TRUE)
               THEN IF (VMDB(TRADS.INDEX3) GT ASEP**2)
                         THEN TRADS.FEATBITS(20) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + BIGVWGT;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_big_vertical_miss_distance;
```

```
--------------------------------------------------------------------------------------

PROCESS feature_biggest_separation_for_negatives;

     <This feature breaks ties among potential resolution advisories with NEGATIVE
      set by determining which advisory set exceeds its 'negative suffices'
      threshold by the largest factor.  QSEP values are used as a final tie-breaker
      if necessary. >

     Maximum ratio = 0;
     Maximum QSEP = 0;

     LOOP;   < Repeat for each maximum-value RA in RADS list. >
          IF (this RA is horizontal)
               THEN ratio = appropriate separation value from HMD matrix /
                             horizontal 'negative suffices' threshold;
               ELSE < RA is vertical. >
                    Ratio = (appropriate separation value from VMDA matrix /
                              vertical 'negative suffices' threshold)**2;
          IF (ratio for this RA GT maximum ratio)
               THEN Set best-RA pointer to point to this RA;
                    Maximum ratio = ratio for this RA;
                    Maximum QSEP = QSEP value for this RA;
          ELSEIF (ratio for this RA EQ maximum ratio)
               THEN IF (QSEP value for this RA GT maximum QSEP)
                         THEN Set best-RA pointer to point to this RA;
                              Maximum QSEP = QSEP value for this RA;
                         ELSE;
          OTHERWISE;
     EXITIF (all maximum-value RAs examined);
     ENDLOOP;

     SET this feature;
     Add this feature's weight to this RADS total value;

END feature_biggest_separation_for_negatives;

----------- RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC ------------
```

```
--------------------------------------------------------------------------------------------

PROCESS feature_biggest_separation_for_negatives;

     FLT (RATIO, VSEP, MAXRATIO, MAXQSEP);

     MAXRATIO = 0;
     MAXQSEP = 0;

     LOOP;   < Repeat for each maximum-value RA in RADS list. >

          IF (TRADS.HORIZ EQ $TRUE)
               THEN RATIO = HMD2(TRADS.INDEX1, TRADS.INDEX2) / MDTHR;
               ELSE IF (RSPND1 EQ $TRUE AND RSPND2 EQ $TRUE)
                         THEN VSEP = VMDA($LEV3);
                         ELSE VSEP = VMDA(TRADS.INDEX3);
                    RATIO = (VSEP / ASEP)**2;

          IF (RATIO GT MAXRATIO)
               THEN Set RADSPTR to point to this RA;
                    MAXRATIO = RATIO;
                    MAXQSEP = QSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3);
          ELSEIF (RATIO EQ MAXRATIO)
               THEN IF (QSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3) GT MAXQSEP)
                         THEN RADSPTR = TRADS;
                              MAXQSEP = QSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3);
                         ELSE;
          OTHERWISE;

     EXITIF (all maximum-value RAs examined);
     ENDLOOP;

     RADSPTR.FEATBITS(25) = $TRUE;;
     RADSPTR.VALUE = RADSPTR.VALUE + BSEPWGT;

END feature_biggest_separation_for_negatives;
```

```
--------------------------------------------------------------------------------
PROCESS feature_biggest_separation_for_positives;

    <This feature breaks ties among potential resolution advisories with NEGATIVE
     not set by choosing the advisory set with the largest PSEP value.
     QSEP values are used as a final tie-breaker if necessary. >

    Maximum PSEP = 0;
    Maximum QSEP = 0;

    LOOP:   < Repeat for each maximum-value RA in RADS list. >

        IF (PSEP value for this RA GT maximum PSEP)
            THEN Set best-RA pointer to point to this RA;
                 Maximum PSEP = PSEP value for this RA;
                 Maximum QSEP = QSEP value for this RA;

        ELSEIF (PSEP value for this RA EQ maximum PSEP)
            THEN IF (QSEP value for this RA GT maximum QSEP)
                      THEN Set best-RA pointer to point to this RA;
                           Maximum QSEP = QSEP value for this RA;
                      ELSE ;

        OTHERWISE :

    EXITIF (all maximum-value RAs examined);
    ENDLOOP;

    SET this feature;
    Add this feature's weight to this RADS total value;

END feature_biggest_separation_for_positives;
```

```
--------------------------------------------------------------------------------------

PROCESS feature_biggest_separation_for_positives;

    FLT (MAXPSEP, MAXQSEP);

    MAXPSEP = 0;
    MAXQSEP = 0;

    LOOP:   < Repeat for each maximum-value RA in RADS list. >

        IF (PSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3) GT MAXPSEP)
            THEN RADSPTR = TRADS;
                 MAXPSEP = PSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3);
                 MAXQSEP = QSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3);

        ELSEIF (PSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3) EQ MAXPSEP)
            THEN IF (QSEP2(TRADS.INDEX1, TRADS.INDEX2, TRADS.INDEX3) GT MAXQSEP)
                     THEN RADSPTR = TRADS;
                          MAXQSEP = QSEP2(TRADS.INDEX1, TRADS.INDEX2,
                                          TRADS.INDEX3);
                     ELSE;

        OTHERWISE:

    EXITIF (all maximum-value RAs examined);
    ENDLOOP;

    SET RADSPTR.FEATBITS(25) = $TRUE;
    RADSPTR.VALUE = RADSPTR.VALUE + BSEPPWGT;

END feature_biggest_separation_for_positives;
```

```
--------------------------------------------------------------------------------------

PROCESS feature_compatible_with_turn;

     <If this advisory set has a horizontal component, attempt to use it to resolve
      the conflict if the horizontal maneuver is compatible with the tracker sensed
      horizontal turn status of each AC.>

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (there is a horizontal component to this advisory set)
               THEN IF (horizontal maneuvers in this set are compatible with
                              turn status for each maneuvered AC)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_compatible_with_turn;
```

```
--------------------------------------------------------------------------------

PROCESS feature_compatible_with_turn;

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (TRADS.HORIZ EQ $TRUE)
               THEN IF ((COMPATTS(ACID1.TURN,TRADS.H1) EQ $TRUE)
                              AND (COMPATTS(ACID2.TURN,TRADS.H2) EQ $TRUE))
                         THEN TRADS.FEATBITS(19) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + CONSTWGT;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_compatible_with_turn;
```

```
--------------------------------------------------------------------------------

PROCESS feature_deliverable;

     <If this is the first time advisories are being selected for this pair, then
      this advisory set may be considered for selection only if it contains negative
      advisories or if it provides greater separation than if the AC were to receive
      no advisories.>

     IF (this is the first time resolution advisories are being selected for
              this pair)
         THEN LOOP;
                  Get the next resolution advisory data set;
              EXITIF (all resolution advisory data sets processed);
                  IF ((separation obtained by responding to this resolution
                           advisory set GT projected separation for neither AC
                           maneuvering) OR (negative resolution advisories are
                           sufficient))
                      THEN SET this feature;
                           Add this feature's weight to this RADS total value;
                           Add this feature's weight to this RADS value used
                                for domino logic processing;
                      ELSE;
              ENDLOOP;

         ELSE LOOP;
                  Get the next RADS;
              EXITIF (done all RADS);
                  SET this feature;
                  Add this feature's weight to this RADS total value;
                  Add this feature's weight to this RADS value used for domino
                       logic processing;
              ENDLOOP;

END feature_deliverable;
```

```
PROCESS feature_deliverable;

    IF ((PREC.ac1.PHMAN EQ $NULLRES) AND (PREC.ac1.PVMAN EQ $NULLRES) AND
            (PREC.ac2.PHMAN EQ $NULLRES) AND (PREC.ac2.PVMAN EQ $NULLRES))
        THEN LOOP:
                Get the next resolution advisory data set;
             EXITIF (all resolution advisory data sets processed):
                IF ((PSEP2(TRADS.INDEX1,TRADS.INDEX2,TRADS.INDEX3) GT
                        PSEP2(2,2,1)) OR (TRADS.NEGATIVE EQ $TRUE))
                    THEN TRADS.FEATBITS(1) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + DELWGT;
                        TRADS.DOMVALUE = TRADS.DOMVALUE + DELWGT;
                    ELSE;
             ENDLOOP;

        ELSE LOOP:
                Get the next resolution advisory data set;
             EXITIF (all resolution advisory data sets processed);
                TRADS.FEATBITS(1) = $TRUE;
                TRADS.VALUE = TRADS.VALUE + DELWGT;
                TRADS.DOMVALUE = TRADS.DOMVALUE + DELWGT;
             ENDLOOP;

END feature_deliverable;
```

```
--------------------------------------------------------------------------
PROCESS feature_dimension_available:

     <Consider this advisory set for selection only if all of its components are
      compatible with the resolution advisories currently being sent to each AC.>

     LOOP:
          Get the next RADS;
     EXITIF (done all RADS);
          SET dimension available feature;
          LOOP:
               Get next AC of subject pair;
          EXITIF (done both aircraft OR dimension available feature not set)
               IF ((the horizontal resolution advisory in this resolution
                         advisory set is compatible with the resolution
                         advisory in the conflict table entry) AND
                         (the vertical resolution advisory in this
                         resolution advisory set is compatible with the
                         resolution advisory in the conflict table entry))
                    THEN;
                    ELSE CLEAR dimension available feature;
          ENDLOOP;

          IF (this feature is set)
               THEN add this feature's weight to this RADS total value;
                    Add this feature's weight to this RADS value used for domino
                         logic processing;
               ELSE;
     ENDLOOP;

END  feature_dimension_available;
```

```
-----------------------------------------------------------------------------

PROCESS feature_dimension_available;

    LOOP:
        Get the next RADS;
    EXITIF (done all RADS);
        TRADS.FEATBITS(2) = $TRUE;

        LOOP:
            Get next AC of subject pair;
        EXITIF (done both aircraft OR TRADS.FEATBITS(2) = $FALSE)
            IF ((COMPAT(TRADS.H,ACID.CTE.HMAN) EQ $TRUE)
                    AND (COMPAT(TRADS.V,ACID.CTE.VMAN) EQ $FALSE))
                THEN:
                ELSE TRADS.FEATBITS(2) = $FALSE;
        ENDLOOP;

        IF (TRADS.FEATBITS(2) EQ $TRUE)
            THEN TRADS.VALUE = TRADS.VALUE + DIMAVWGT;
                TRADS.DOMVALUE = TRADS.DOMVALUE + DIMAVWGT;
            ELSE:
    ENDLOOP;

END  feature_dimension_available;
```

```
--------------------------------------------------------------------------------
PROCESS feature_domino_logic;
    <Determine if an advisory to an AC causes that AC to come in conflict with
     another AC.  Evaluate only those advisories tied for selection based on
     features with higher priority than the domino features.  Two features
     are used to evaluate the domino logic: one feature is set if this
     advisory causes a domino conflict for neither subject AC; the other is set
     if a domino conflict is caused for only one of the subject AC.>
    LOOP:
        Get next AC of subject pair;
    EXITIF (both AC have been processed);
        IF (this AC is maneuvered)
            THEN LOOP:
                     Get next potential resolution advisory from RADS;
                 EXITIF (no more potential resolution advisories left);
                     IF (value of features down to domino) EQ (value of
                               maximum valued potential resolution advisory))
                         THEN IF (neither of the domino features is set)
                                  THEN SET 'neither AC domino' feature;
                              PERFORM domino_conflict_detection;
                              IF (domino conflict detected)
                                  THEN IF ('domino conflict created for one AC'
                                               feature is set)
                                           THEN CLEAR 'domino conflict created
                                                    for one AC' feature;
                                           ELSE CLEAR 'domino conflict created
                                                    for neither AC' feature;
                                                SET 'domino conflict created
                                                    for one AC' feature;
                                  ELSE;
                         ELSE;
                 ENDLOOP;
                 PERFORM domino_features_weight_addition;
            ELSE;
    ENDLOOP;
END feature_domino_logic;
-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  ------------
```

```
---------------------------------------------------------------------------------
PROCESS feature_domino_logic;

    LOOP:
        Get next AC of subject pair;
    EXITIF (both AC have been processed);
        IF (RSPND = $TRUE)
            THEN LOOP:
                    Get next potential RA from RADS;
                 EXITIF (no more potential RAS left);
                    IF (TRADS.DOMVALUE EQ MAXVALUE)
                        THEN IF ((TRADS.FEATBITS(7) EQ $FALSE) AND
                                 (TRADS.FEATBITS(8) EQ $FALSE))
                                THEN TRADS.FEATBITS(7) = $TRUE;
                                ELSE;

                             PERFORM domino_conflict_detection;
                             IF (potential RA domino status variable EQ $DOMCC)
                                THEN IF (TRADS.FEATBITS(8) EQ $TRUE)
                                        THEN TRADS.FEATBITS(8) = $FALSE;
                                        ELSE TRADS.FEATBITS(7) = $FALSE;
                                             TRADS.FEATBITS(8) = $TRUE;
                                ELSE;
                        ELSE;
                 ENDLOOP;

                 PERFORM domino_features_weight_addition;
            ELSE;
    ENDLOOP;

END feature_domino_logic;
```

```
-------------------------------------------------------------------------------

PROCESS feature_fast_unmaneuvered_slow_maneuvered;

     <If one AC is unmaneuvered, and that AC has a large vertical velocity, its
      speed is much greater than the speed of the maneuvered AC, and the AC are
      approximately head-on, then attempt to resolve this conflict with a double
      dimension advisory set.>

     IF (either AC is not maneuvered)
          THEN compute track crossing angle;
               Calculate ratio of squared speed of uncmded AC to squared
                    speed of cmded AC;
               IF ((uncmded AC has a dangerous vertical velocity) AND
                         (speed ratio GT a threshold) AND
                         (track crossing angle is between certain limits
                         approximately head-on))
                    THEN LOOP;
                              Get the next resolution advisory data set;
                         EXITIF (all resolution advisory data sets processed);
                              IF (this is a double dimension resolution advisory
                                        set)
                                   THEN SET this feature;
                                        Add this feature's weight to this RADS total
                                             value;
                                   ELSE;
                         ENDLOOP;
                    ELSE;
          ELSE;

END feature_fast_unmaneuvered_slow_maneuvered;
```

```
---------------------------------------------------------------------------------------
PROCESS feature_fast_unmaneuvered_slow_maneuvered;

     FLT TRATIO;

     IF ((RSPND1 EQ $FALSE) OR (RSPND2 EQ $FALSE))
          THEN compute track crossing angle;   <TXTH>
               IF (RSPND1 EQ $FALSE)
                    THEN TRATIO = ACID1.VSQ / ACID2.VSQ;
                         IF (ACID1.ZD GT ZDTH)
                              THEN FSTUNCZD = $TRUE;
                              ELSE FSTUNCZD = $FALSE;
                    ELSE TRATIO = ACID2.VSQ / ACID1.VSQ;
                         IF (ACID2.ZD GT ZDTH)
                              THEN FSTUNCZD = $TRUE;
                              ELSE FSTUNCZD = $FALSE;

               IF ((FSTUNCZD EQ $TRUE) AND (TRATIO GT VRATIO) AND
                         ((TXTH1 LT TXTH) AND (TXTH LT TXTH2)))
                    THEN LOOP;
                              Get the next resolution advisory data set;
                         EXITIF (all resolution advisory data sets processed);
                              IF (TRADS.SINGLE EQ $FALSE)
                                   THEN TRADS.FEATBITS(12) = $TRUE;
                                        TRADS.VALUE = TRADS.VALUE + FUCSCWGT;
                                   ELSE;
                         ENDLOOP;
                    ELSE;
          ELSE;

END feature_fast_unmaneuvered_slow_maneuvered;
```

```
--------------------------------------------------------------------------------------
PROCESS feature_initial_resolution_advisory_selection;

    <If the calling task requested single dimension advisories, attempt to select
     single dimension advisories.>

    IF (single dimension resolution advisories are requested by calling routine)
        THEN LOOP:
                  Get the next resolution advisory data set;
             EXITIF (all resolution advisory data sets processed);
                  IF (this resolution advisory set is single dimension)
                      THEN SET this feature;
                           Add this feature's weight to this RADS total value;
                      ELSE;
             ENDLOOP;
        ELSE;

END feature_initial_resolution_advisory_selection;
```

```
--------------------------------------------------------------------------------
PROCESS feature_initial_resolution_advisory_selection;

    IF (SNGDIN EQ $TRUE)
        THEN LOOP;
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF (TRADS.SINGLE EQ $TRUE)
                    THEN TRADS.FEATBITS(17) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + SNGLDWGT;
                    ELSE;
            ENDLOOP;
        ELSE;

END feature_initial_resolution_advisory_selection;
```

```
--------------------------------------------------------------------------------
PROCESS feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;

     <An aircraft in the current conflict pair is in another conflict pair, for
      which it was modeled as unmaneuvered.  Calculate the separation achieved
      for the previous conflict pair, based on the subject AC being maneuvered.>

     SET maneuvered_unmaneuvered conflict feature;
     LOOP:
          Get next AC from the subject pair);
     EXITIF (both AC processed OR maneuvered_unmaneuvered conflict feature not set);
          IF (this AC is maneuvered)
               THEN IF (this is a double dimension resolution advisory)
                         THEN PERFORM multi_AC_vertical_maneuvered_unmaneuvered_
                                                       conflict_determination;
                              IF (maneuvered_unmaneuvered conflict feature set)
                                   THEN PERFORM multi_AC_horizontal_maneuvered_
                                                 unmaneuvered_conflict_determination;
                                   ELSE;

                         ELSE IF (vertical resolution advisory in this resolution
                                        advisory set)
                                   THEN PERFORM multi_AC_vertical_maneuvered_
                                                 unmaneuvered_conflict_determination;
                                   ELSE PERFORM multi_AC_horizontal_maneuvered_
                                                 unmaneuvered_conflict_determination;
               ELSE;
     ENDLOOP;

     IF (this feature is set)
          THEN add this feature's weight to this RADS total value;
               Add this feature's weight to this RADS value used for domino logic
                    processing;
          ELSE:

END feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;
-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  -------------
```

```
--------------------------------------------------------------------------------------

PROCESS feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;

     TRADS.FEATBITS(3) = $TRUE;
     LOOP:
          Get next AC from the subject pair);
     EXITIF (both AC processed OR (TRADS.FEATBITS(3) EQ $FALSE));
          IF (RSPND EQ $TRUE)
               THEN IF (TRADS.SINGLE EQ $FALSE)
                         THEN PERFORM multi_AC_vertical_maneuvered_
                                             unmaneuvered_conflict_determination;
                              <for the vertical portion of the RA>
                              IF (TRADS.FEATBITS(3) EQ $TRUE)
                                   THEN PERFORM multi_AC_horizontal_maneuvered_
                                             unmaneuvered_conflict_determination;
                                   <for horiz portion of the RA>
                                   ELSE;

                         ELSE IF (TRADS.VERT EQ $TRUE)
                                   THEN PERFORM multi_AC_vertical_maneuvered_
                                            unmaneuvered_conflict_determination;
                                   ELSE PERFORM multi_AC_horizontal_maneuvered_
                                            unmaneuvered_conflict_determination;
               ELSE;
     ENDLOOP;

     IF (TRADS.FEATBITS(3) EQ $TRUE)
          THEN TRADS.VALUE = TRADS.VALUE + UNMANWGT;
               TRADS.DOMVALUE = TRADS.DOMVALUE + UNMANWGT;
          ELSE;

END feature_maneuvered_unmaneuvered_conflict_multi_AC_definition;
```

```
------------------------------------------------------------------------------
PROCESS feature_maneuvered_unmaneuvered_conflict_two_AC_definition;
     <The Two-aircraft Resolution logic definition of the maneuvered_unmaneuvered
      conflict feature checks for previous conflict pairs in which the currently
      maneuvered subject AC was modeled as unmaneuvered.  For the current conflict,
      do not use advisories in the dimension that was used to resolve the
      previous conflict.>

     LOOP:
          Get the next RADS;
     EXITIF (done all RADS);
          SET maneuvered_unmaneuvered conflict feature;
          LOOP:
               Get next AC of subject pair;
          EXITIF (both AC processed OR maneuvered_unmaneuvered conflict feature not
                    set);
               IF (this AC is maneuvered)
                    THEN IF (there is a vertical component to this advisory set)
                              THEN PERFORM two_AC_vertical_maneuvered_unmaneuvered_
                                                                 conflict_determination;
                              ELSE;
                         IF (maneuvered_unmaneuvered conflict feature favored AND
                                   there is a horizontal component to this resolution
                                   advisory)
                              THEN PERFORM two_AC_horizontal_maneuvered_
                                                       unmaneuvered_conflict_determination;
                              ELSE;
                    ELSE;
          ENDLOOP;
          IF (this feature is set)
               THEN add this feature's weight to this RADS total value;
                    Add this feature's weight to this RADS value used for domino
                         logic processing;
               ELSE;
     ENDLOOP;
END feature_maneuvered_unmaneuvered_conflict_two_AC_definition;
----------- RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC -------------
```

```
PROCESS feature_maneuvered_unmaneuvered_conflict_two_AC_definition;

     <The Two-aircraft Resolution logic definition of the maneuvered_unmaneuvered
      conflict feature checks for previous conflict pairs in which the currently
      maneuvered subject AC is unmaneuvered.  If this condition exists, this
      feature is not set.>

     LOOP:
          Get next RADS;
     EXITIF (done all RADS);
          TRADS.FEATBITS(3) = $TRUE;
          LOOP:
               Get next AC of subject pair;
          EXITIF (both AC processed OR TRADS.FEATBITS(3) EQ $FALSE);
               IF (RSPND EQ $TRUE)
                    THEN IF (TRADS.VERT EQ $TRUE)
                              THEN PERFORM two_AC_vertical_maneuvered_unmaneuvered_
                                                            conflict_determination;
                              ELSE;
                         IF ((TRADS.FEATBITS(3) EQ $TRUE) AND (TRADS.HORIZ EQ $TRUE))
                              THEN PERFORM two_AC_horizontal_maneuvered_
                                             unmaneuvered_conflict_determination;
                              ELSE;
                    ELSE;
          ENDLOOP;

          IF (TRADS.FEATBITS(3) EQ $TRUE)
               THEN TRADS.VALUE = TRADS.VALUE + UNMANWGT;
                    TRADS.DONVALUE = TRADS.DONVALUE + UNMANWGT;
               ELSE:
     ENDLOOP;

END feature_maneuvered_unmaneuvered_conflict_two_AC_definition;
```

```
PROCESS feature_negative_resolution_advisories_do_not_reverse_maneuver;

     <Attempt to select negative horizontal advisories that are compatible with the
      tracker sensed turn status or negative vertical advisories that are compatible
      with the vertical velocity direction of the AC.>

     LOOP;
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (negative resolution advisories are selected)
               THEN IF (there is a horizontal component to this set)
                         THEN IF (each maneuvered AC's horizontal turn status is
                                        compatible with this resolution advisory
                                        set)
                              THEN SET this feature;
                                   Add this feature's weight to this RADS total
                                        value;
                              ELSE;
                    ELSE IF (each maneuvered AC's vertical velocity is
                                   compatible with this resolution
                                   advisory set)
                              THEN SET this feature;
                                   Add this feature's weight to this
                                        RADS total value;
                              ELSE;
               ELSE;
     ENDLOOP;

END feature_negative_resolution_advisories_do_not_reverse_maneuver;
```

```
--------------------------------------------------------------------------------
PROCESS feature_negative_resolution_advisories_do_not_reverse_maneuver;

    LOOP;
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (TRADS.NEGATIVE EQ $TRUE)
            THEN IF (TRADS.HORIZ EQ $TRUE)
                    THEN IF ((COMPATTS(ACID1.TURN,TRADS.H1) EQ $TRUE) AND
                             (COMPATTS(ACID2.TURN,TRADS.H2) EQ $TRUE))
                        THEN TRADS.FEATBITS(11) = $TRUE;
                             TRADS.VALUE = TRADS.VALUE + NDNRMWGT;
                        ELSE;
                    ELSE IF ((COMPATZD(ACID1.ZD,TRADS.V1) EQ $TRUE) AND
                             (COMPATZD(ACID2.ZD,TRADS.V2) EQ $TRUE))
                        THEN TRADS.FEATBITS(11) = $TRUE;
                             TRADS.VALUE = TRADS.VALUE + NDNRMWGT;
                        ELSE;
            ELSE;
    ENDLOOP;

END feature_negative_resolution_advisories_do_not_reverse_maneuver;
```

```
--------------------------------------------------------------------------------

PROCESS feature_negative_resolution_advisories_suffice;

     <Attempt to use negative resolution advisories to resolve the conflict.>

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (negative resolution advisories are sufficient)
               THEN SET this feature;
                    Add this feature's weight to this RADS total value;
               ELSE:
     ENDLOOP;

END feature_negative_resolution_advisories_suffice;
```

```
------------------------------------------------------------------------------------------

PROCESS feature_negative_resolution_advisories_suffice;

    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (TRADS.NEGATIVE EQ $TRUE)
            THEN TRADS.FEATBITS(10) = $TRUE;
                TRADS.VALUE = TRADS.VALUE + NEGSFWGT;
            ELSE;
    ENDLOOP;

END feature_negative_resolution_advisories_suffice;
```

```
--------------------------------------------------------------------------------
PROCESS feature_no_level_off_time_for_verticals;

    <If the AC do not have time to level-off vertically before they cross altitude,
     attempt to use an advisory set with a horizontal component to resolve this
     conflict.>

    IF ((vertical tau GT time to vertical crossover) AND
              (vertical tau LT time to level-off))
        THEN LOOP:
                  Get the next resolution advisory data set;
             EXITIF (all resolution advisory data sets processed);
                  IF (there is a horizontal component to this resolution advisory)
                      THEN SET this feature;
                           Add this feature's weight to this RADS total value;
                      ELSE;
             ENDLOOP;
        ELSE;


END feature_no_level_off_time_for_verticals;
```

```
--------------------------------------------------------------------------------
PROCESS feature_no_level_off_time_for_verticals;

     IF ((TV1 LT ELENTRY.TV) AND (ELENTRY.TV LT TV2))
          THEN LOOP;
                    Get the next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (TRADS.HORIZ EQ STRUE)
                         THEN TRADS.FEATBITS(14) = STRUE;
                              TRADS.VALUE = TRADS.VALUE + NOLEVWGT;
                         ELSE;
               ENDLOOP;
          ELSE;

END feature_no_level_off_time_for_verticals;
```

```
------------------------------------------------------------------------------------------

PROCESS feature_non_response_to_positive_resolution_advisories_detected;

     <If the Master Resolution Task has detected that the AC are not responding to
      single dimension positive advisories, attempt to resolve the conflict with
      double dimension (positive) advisories.>

     IF (double dimension resolution advisories are requested by calling routine)
          THEN LOOP;
                         Get the next resolution advisory data set;
                    EXITIF (all resolution advisory data sets processed);
                         IF (this is a double dimension resolution advisory set)
                              THEN SET this feature;
                                   Add this feature's weight to this RADS total value;
                              ELSE;
                    ENDLOOP;
          ELSE;

END feature_non_response_to_positive_resolution_advisories_detected;
```

```
--------------------------------------------------------------------------------

PROCESS feature_non_response_to_positive_resolution_advisories_detected;

    IF (SNGDIM EQ $FALSE)
        THEN LOOP;
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF (TRADS.SINGLE EQ $FALSE)
                    THEN TRADS.FEATBITS(15) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + NRESPWGT;
                    ELSE;
            ENDLOOP;
        ELSE;

END feature_non_response_to_positive_resolution_advisories_detected;
```

```
------------------------------------------------------------------------------
PROCESS feature_PSEP_GE_SEP1;

     <Attempt to use advisories that provide at least a minimum amount of
      separation.>

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (predicted separation for response to this resolution advisory set GE
                   separation threshold)
               THEN SET this feature;
                    Add this feature's weight to this RADS total value;
                    Add this feature's weight to this RADS value used for domino
                         logic processing;
               ELSE;
     ENDLOOP;

END feature_PSEP_GE_SEP1;
```

```
--------------------------------------------------------------------------------

PROCESS feature_PSEP_GE_SEP1;

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (PSEP?(TRADS.INDEX1,TRADS.INDEX2,TRADS.INDEX3) GE RESADV.SEP1)
               THEN TRADS.FEATBITS(4) = $TRUE;
                    TRADS.VALUE = TRADS.VALUE + PSEP1WGT;
                    TRADS.DOMVALUE = TRADS.DOMVALUE + PSEP1WGT;
               ELSE;
     ENDLOOP;

END feature_PSEP_GE_SEP1;
```

```
--------------------------------------------------------------------------------

PROCESS feature_PSEP_GE_SEP2;

    CLEAR maximum separation for single dimension advisory sets;
    CLEAR maximum separation for double dimension advisory sets;
    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all reslution advisory data sets processed);
        IF (all absolute features are set for this RADS)
            THEN IF (this is a single dimension RADS)
                    THEN save maximum of this RADS predicted separation and
                            previously saved maximum for single dimension
                            advisory sets;
                    ELSE save maximum of this RADS predicted separation and
                            previously saved maximum for double dimension
                            advisory sets;
            ELSE;
    ENDLOOP;
    SET separation threshold for single dimension advisories to maximum of
        minimum acceptable threshold and a percentage of the maximum
        separation saved for single dimension advisory sets;
    SET separation threshold for double dimension advisories to maximum of
        minimum acceptable threshold and a percentage of the maximum
        separation saved for double dimension advisory sets;

    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (predicted separation for response to this resolution advisory set GE
                separation threshold)
            THEN SET this feature;
                Add this feature's weight to this RADS total value;
            ELSE;
    ENDLOOP;

END feature_PSEP_GE_SEP2;
```

```
-----------------------------------------------------------------------------

PROCESS feature_PSEP_GE_SEP2;
     FLT (MAXSEPS, MAXSEPD, SEP2S, SEP2D);
     MAXSEPS = 0;
     MAXSEPD = 0;
     LOOP;
          Get the next RADS;
     EXITIF (all RADS processed);
          IF ((TRADS.FEATBITS(1) EQ $TRUE) AND (TRADS.FEATBITS(2) EQ $TRUE) AND
                    (TRADS.FEATBITS(3) EQ $TRUE))
               THEN IF (TRADS.SINGLE EQ $TRUE)
                         THEN MAXSEPS = MAX(MAXSEPS,PSEP2(TRADS.INDEX1,TRADS.INDEX2,
                                   TRADS.INDEX3));
                         ELSE MAXSEPD = MAX(MAXSEPD,PSEP2(TRADS.INDEX1,TRADS.INDEX2,
                                   TRADS.INDEX3));
               ELSE;
     ENDLOOP;
     SEP2S = SEP2AP * MAXSEPS;
     SEP2D = SEP2AP * MAXSEPD;
     SEP2S = MAX(RESADV.SEP1,SEP2S);
     SEP2D = MAX(RESADV.SEP1,SEP2D);

     LOOP;
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (TRADS.SINGLE EQ $TRUE)
               THEN IF (PSEP2(TRADS.INDEX1,TRADS.INDEX2,TRADS.INDEX3) GE SEP2S)
                         THEN TRADS.FEATBITS(18) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + PSEP2WGT;
                         ELSE;
               ELSE IF (PSEP2(TRADS.INDEX1,TRADS.INDEX2,TRADS.INDEX3) GE SEP2D)
                         THEN TRADS.FEATBITS(18) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + PSEP2WGT;
                         ELSE;
     ENDLOOP;
END feature_PSEP_GE_SEP2;
------------  RESOLUTION ADVISORIES EVALUATION ROUTINE LOW-LEVEL LOGIC  ------------
```

```
-------------------------------------------------------------------------------------------

PROCESS feature_reinforce_res_adv_from_non_connected_site_or_BCAS;

     <Determine if either maneuvered AC has a resolution advisory from another ATARS
      site or from BCAS.  If so, attempt to use an advisory set that reinforces the
      advisories from the other source.>

     PERFORM other_sources_resolution_advisory_determination;
     IF (either AC is receiving a resolution advisory from a non-connected site OR
               BCAS)
          THEN LOOP:
                    Get the next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (the resolution advisories in this set reinforce the
                              resolution advisories from the other source)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                              Add this feature's weight to this RADS value used
                                   for domino logic processing;
                         ELSE;
               ENDLOOP;
          ELSE;

END feature_reinforce_res_adv_from_non_connected_site_or_BCAS;
```

```
--------------------------------------------------------------------------------

PROCESS feature_reinforce_res_adv_from_non_connected_site_or_BCAS;

     PERFORM other_sources_resolution_advisory_determination;
     IF ((OSHMAN1 NE $NULLRES) OR (OSHMAN2 NE $NULLRES)
                OR (OSVMAN1 NE $NULLRES) OR (OSVMAN2 NE $NULLRES))
          THEN LOOP;
                     Get the next resolution advisory data set;
                EXITIF (all resolution advisory data sets processed);
                     IF ((REINF(TRADS.H1,OSHMAN1) EQ $TRUE) OR
                               (REINF(TRADS.H2,OSHMAN2) EQ $TRUE) OR
                               (REINF(TRADS.V1,OSVMAN1) EQ $TRUE) OR
                               (REINF(TRADS.V2,OTSVMAN2) EQ $TRUE))
                          THEN TRADS.FEATBITS(5) = $TRUE;
                               TRADS.VALUE = TRADS.VALUE + OTHSTWGT;
                               TRADS.DOMVALUE = TRADS.DOMVALUE + OTHSTWGT;
                          ELSE;
                ENDLOOP;
          ELSE;

END feature_reinforce_res_adv_from_non_connected_site_or_BCAS;
```

```
--------------------------------------------------------------------------------
PROCESS feature_reinforces_prior_resolution_advisories;

    <Attempt to use an advisory set that reinforces the advisories selected
     previously for this pair.>

    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF (any resolution advisory in this set reinforces a resolution
                advisory in the pair record))
            THEN SET this feature;
                Add this feature's weight to this RADS total value;
            ELSE;
    ENDLOOP;

END feature_reinforces_prior_resolution_advisories;
```

```
--------------------------------------------------------------------------------
PROCESS feature_reinforces_prior_resolution_advisories;

    LOOP:
        Get the next resolution advisory data set;
    EXITIF (all resolution advisory data sets processed);
        IF ((REINF(TRADS.H1,PREC.ac1.PHMAN) EQ $TRUE) OR
                (REINF(TRADS.H2,PREC.ac2.PHMAN) EQ $TRUE) OR
                (REINF(TRADS.V1,PREC.ac1.PVMAN) EQ $TRUE) OR
                (REINF(TRADS.V2,PREC.ac2.PVMAN) EQ $TRUE)))
            THEN TRADS.FEATBITS(22) = $TRUE;
                TRADS.VALUE = TRADS.VALUE + REPRAWGT;
            ELSE;
    ENDLOOP;

END feature_reinforces_prior_resolution_advisories;
```

```
------------------------------------------------------------------------------------

PROCESS feature_reinforces_turn;

     <Attempt to select an advisory set with a horizontal component that reinforces
      a tracker sensed turn.>

     LOOP;
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (there are horizontal components to this set)
               THEN IF (any horizontal resolution advisory reinforces a tracker
                              sensed turn)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_reinforces_turn;
```

```
-------------------------------------------------------------------------------------
PROCESS feature_reinforces_turn;

     LOOP:
          Get the next resolution advisory data set;
     EXITIF (all resolution advisory data sets processed);
          IF (TRADS.HORIZ EQ $TRUE)
               THEN IF (((ACID1.TURN EQ $STRNGRGT) AND
                              (TRADS.H1 EQ $TR)) OR
                              ((ACID1.TURN EQ $STRNGLFT) AND
                              (TRADS.H1 EQ $TL)) OR
                              ((ACID2.TURN EQ $STRNGRGT) AND
                              (TRADS.H2 EQ $TR)) OR
                              ((ACID2.TURN EQ $STRNGLFT) AND
                              (TRADS.H2 EQ $TL)))
                         THEN TRADS.FEATBITS(24) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + REINTWGT;
                         ELSE;
               ELSE;
     ENDLOOP;

END feature_reinforces_turn;
```

```
--------------------------------------------------------------------------------
PROCESS feature_speed_check;

     <If a maneuvered AC has a large velocity, attempt to resolve the conflict with
      an advisory set containing a vertical component.  Otherwise, if all
      maneuvering AC have a small velocity, attempt to use an advisory set with a
      horizontal component.>

     IF (a maneuvered AC has a horizontal velocity that is considered fast)
          THEN LOOP;
                    Get the next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (this resolution advisory set has a vertical component)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                         ELSE;
               ENDLOOP;

          ELSE IF (all maneuvering AC have a horizontal velocity that is
                         considered slow)
                    THEN LOOP;
                              Get the next resolution advisory data set;
                         EXITIF (all resolution advisory data sets processed);
                              IF (this resolution advisory set has a horizontal
                                        component)
                                   THEN SET this feature;
                                        Add this feature's weight to this RADS total
                                             value;
                                   ELSE;
                         ENDLOOP;
                    ELSE;

END feature_speed_check;
```

```
--------------------------------------------------------------------------------
PROCESS feature_speed_check;

     IF (((RSPND1 EQ $TRUE) AND (ACID1.VSQ GT VFASTSQ)) OR
              ((RSPND2 EQ $TRUE) AND (ACID2.VSQ GT VFASTSQ)))
          THEN LOOP:
                    Get the next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (TRADS.VERT EQ $TRUE)
                         THEN TRADS.FEATBITS(23) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + SPDCKWGT;
                         ELSE;
               ENDLOOP;

          ELSE IF (((RSPND1 EQ $FALSE) OR (ACID2.VSQ LT VSLOWSQ)) AND
                         ((RSPND2 EQ $FALSE) OR (ACID2.VSQ LT VSLOWSQ)))
                    THEN LOOP:
                              Get the next resolution advisory data set;
                         EXITIF (all resolution advisory data sets processed);
                              IF (TRADS.HORIZ EQ $TRUE)
                                   THEN TRADS.FEATBITS(23) = $TRUE;
                                        TRADS.VALUE = TRADS.VALUE + SPDCKWGT;
                                   ELSE;
                         ENDLOOP;
                    ELSE;

END feature_speed_check;
```

```
-------------------------------------------------------------------------------

PROCESS feature_terrain_or_obstacle_alert;

     <If either AC is receiving a terrain or obstacle alert, attempt to resolve the
      conflict with horizontal-only advisories.>

     IF (a terrain or obstacle avoidance warning is being given)
          THEN LOOP:
                         Get the next resolution advisory data set;
               EXITIF (all resolution advisory data sets processed);
                    IF (this is a horizontal dimension only resolution advisory set)
                         THEN SET this feature;
                              Add this feature's weight to this RADS total value;
                              Add this feature's weight to this RADS value used
                                   for domino logic processing;
                         ELSE;
               ENDLOOP;
          ELSE;

END feature_terrain_or_obstacle_alert;
```

```
--------------------------------------------------------------------------------
PROCESS feature_terrain_or_obstacle_alert;

    IF (either AC is receiving a terrain or obstacle alert)
        THEN LOOP:
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF ((TRADS.SINGLE EQ $TRUE) AND (TRADS.HORIZ EQ $TRUE))
                    THEN TRADS.FEATBITS(6) = $TRUE;
                        TRADS.VALUE = TRADS.VALUE + TEROBWGT;
                        TRADS.DOWVALUE = TRADS.DOWVALUE + TEROBWGT;
                    ELSE:
            ENDLOOP:
        ELSE:

END feature_terrain_or_obstacle_alert;
```

```
-----------------------------------------------------------------------------

PROCESS feature_unmaneuvered_with_large_vertical_rate;

    <If an unmaneuvered AC has a dangerous (large) vertical velocity, attempt to
     resolve the conflict with an advisory set that has a horizontal component.>

    IF (an uncmded AC has a large vertical rate)
        THEN LOOP:
                    Get the next resolution advisory data set;
                EXITIF (all resolution advisory data sets processed);
                    IF (this resolution advisory set has a horizontal component)
                        THEN SET this feature;
                            Add this feature's weight to this RADS total value;
                        ELSE;
                ENDLOOP;
        ELSE;

END feature_unmaneuvered_with_large_vertical_rate;
```

```
PROCESS feature_unmaneuvered_with_large_vertical_rate;

    IF (((RSPND1 EQ $FALSE) AND (ACID1.ZD GT ZDTH)) OR
            ((RSPND2 EQ $FALSE) AND (ACID2.ZD GT ZDTH)))
        THEN LOOP;
                Get the next resolution advisory data set;
            EXITIF (all resolution advisory data sets processed);
                IF (TRADS.HORIZ EQ $TRUE)
                    THEN TRADS.FEATBITS(13) = $TRUE;
                         TRADS.VALUE = TRADS.VALUE + UCLVRWGT;
                    ELSE;
            ENDLOOP;
        ELSE;

END feature_unmaneuvered_with_large_vertical_rate;
```

```
--------------------------------------------------------------------------------
PROCESS highest_valued_potential_resolution_advisory_sets_count;

    <Determine the maximum value of those RADS with all absolute features
     set.  Only the features with higher priority than the domino features are
     considered.  Also, count the number of RADS whose value is equal to the
     maximum value.>

    SET value of highest valued RADS to zero;
    SET number of maximum valued RADS to zero;
    SET pointer to selected RADS to null;

    LOOP:
        Get next RADS;
    EXITIF (processed every RADS);
        IF (all absolute features set for this RADS)
            THEN IF (value of features down to domino GT
                          value of maximum valued RADS)
                     THEN SET value of maximum valued RADS to the value
                              of this RADS features down to domino;
                          SET number of maximum valued RADS to one;
                          SET selected resolution advisory pointer to this RADS;
                 ELSEIF (value of features down to domino EQ
                          value of highest valued RADS)
                     THEN increment number of maximum valued RADS;
                 OTHERWISE;
            ELSE;
    ENDLOOP;

END highest_valued_potential_resolution_advisory_sets_count;
```

```
--------------------------------------------------------------------------------

PROCESS highest_valued_potential_resolution_advisory_sets_count;

     MAXVALUE = 0;
     NUMPRA = 0;
     RADSPTR = $NULL;

     LOOP:
          Get next potential RA set from the RADS;
     EXITIF (processed every potential RA set);
          IF ((TRADS.FEATBITS(1) EQ $TRUE) AND (TRADS.FEATBITS(2) EQ $TRUE)
                    AND (TRADS.FEATBITS(3) EQ $TRUE))
               THEN IF (TRADS.DOMVALUE GT MAXVALUE)
                         THEN MAXVALUE = TRADS.DOMVALUE;
                              NUMPRA = 1;
                              RADSPTR = TRADS;
                    ELSEIF (TRADS.DOMVALUE EQ MAXVALUE)
                              THEN NUMPRA = NUMPRA + 1;
                    OTHERWISE;
               ELSE;
     ENDLOOP;

END highest_valued_potential_resolution_advisory_sets_count;
```

```
-----------------------------------------------------------------------------
PROCESS multi_AC_conflict_possible_resolution_advisories;

    <Determine the possible resolution advisories for the subject conflict pair.
     If the AC are close in altitude, choose opposite sense vertical advisories.
     Otherwise choose same sense vertical advisories.>

    Calculate the vertical separation of the pair after 8 seconds;
    IF (vertical separation is close)
         THEN select vertical resolution advisories for both AC opposite to
                   those selected by the two-AC resolution logic;
         ELSE select CLIMB for both AC;
    CALL PSEP_MATRIX_GENERATOR;

    IF (same sense vertical resolution advisories selected for both AC)
         THEN select DESCEND for both AC;
              CALL PSEP_MATRIX_GENERATOR;
         ELSE;

END multi_AC_conflict_possible_resolution_advisories;
```

```
------------------------------------------------------------------------------------------

PROCESS multi_AC_conflict_possible_resolution_advisories;

     Z8SEC1 = ACID1.Z + ACID1.ZD * TVRULE;
     Z8SEC2 = ACID2.Z + ACID2.ZD * TVRULE;
     IF ((Z8SEC1 - Z8SEC2) LT ZCARE)
          THEN TVERT = VERTRA1;
               VERTRA1 = VERTRA2;
               VERTRA2 = TVERT;
          ELSE VERTRA1 = $CL;
               VERTRA2 = $CL;

     CALL PSEP_MATRIX_GENERATOR
             IN (ACID1, ACID2, RSPND1, RSPND2, VERTRA1, VERTRA2)
             OUT (TRADS.MATPTR, RAPP1, RAPP2);

     IF (VERTRA1 EQ VERTRA2)
          THEN VERTRA1 = $DES;
               VERTRA2 = $DES;
               CALL PSEP_MATRIX_GENERATOR
                        IN (ACID1, ACID2, RSPND1, RSPND2, VERTRA1, VERTRA2)
                        OUT (TRADS.MATPTR, RAPP1, RAPP2);
          ELSE;

END multi_AC_conflict_possible_resolution_advisories;
```

```
--------------------------------------------------------------------------------
PROCESS multi_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;

     <If an AC that is maneuvered for the current conflict is unmaneuvered in
      another conflict that is being resolved using a horizontal advisory, then
      evaluate how the horizontal advisory for the current conflict will affect
      the previous conflicts' resolution.  If the predicted separation for the
      previous conflict is projected to be less than a minimum acceptable
      separation, do not use this advisory set for the current conflict.>

     IF (a maneuvered AC is unmaneuvered in another conflict AND
                   resolution for the previous conflict is in the same
                   dimension as this potential resolution advisory)
          THEN save horizontal resolution advisory from the RADS for the subject AC;
               Save pointers to subject AC and AC in previous conflict with
                    subject AC;
               CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION;
               IF (predicted 3-D separation for previous conflict LT
                         minimum acceptable 3-D separation threshold)
                    THEN CLEAR maneuvered_unmaneuvered conflict feature;
                    ELSE;
          ELSE;

END multi_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;
```

```
------------------------------------------------------------------------------------

PROCESS multi_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;

    FLT PSEP;

    IF (a maneuvered AC is unmaneuvered in another conflict AND
            resolution for the previous conflict is in the horizontal dimension)
        THEN save horizontal RA from the RADS for the subject AC;
            Save pointers to subject AC and AC in previous conflict with
                subject AC;
            CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION
                IN (RAs for two AC, AC state vectors)
                OUT (PSEP);
            IF (PSEP LT RESADV.SEP1)
                THEN TRADS.FEATBITS(3) = $FALSE;
                ELSE;
        ELSE;

END multi_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;
```

```
--------------------------------------------------------------------------------
PROCESS multi_AC_resolution_logic_advisories_calculations;

    <Evaluate the absolute features for the new advisory sets, using the
     multi-AC logic definition of the features.>

    LOOP;
         Get next RADS;
    EXITIF (all RADS processed);
         PERFORM absolute_features_evaluation_multi_AC_resolution_definition;
         IF (all absolute features set)
              THEN increment count of RADS with all absolute features set;
                   IF (pointer to selected RADS is null)
                        THEN save a pointer to the selected advisory set;
                        ELSE;
              ELSE;
    ENDLOOP;

END mult_AC_resolution_logic_advisories_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS multi_AC_resolution_logic_advisories_calculations;

     CALL PSEP_MATRIX_GENERATOR
             IN (ACID1, ACID2, RSPND1, RSPND2, VERTRA1, VERTRA2)
             OUT (RADS.MATPTR, RAPP1, RAPP2);
     IF ((VERTRA1 EQ SCL) AND (VERTRA2 EQ SCL))
          THEN VERTRA1 = SDES;
               VERTRA2 = SDES;
     CALL PSEP_MATRIX_GENERATOR
             IN (ACID1, ACID2, RSPND1, RSPND2, VERTRA1, VERTRA2)
             OUT (RADS.MATPTR, RAPP1, RAPP2);
     ELSE;

     LOOP;
          Get next RADS;
     EXITIF (all RADS processed);
          PERFORM absolute_features_evaluation_multi_AC_resolution_definition;
          IF ((TRADS.FEATBITS(1) EQ STRUE) AND
                    (TRADS.FEATBITS(2) EQ STRUE) AND (TRADS.FEATBITS(3) EQ STRUE))
               THEN NPRAABS = NPRAABS + 1;
                    IF (RADSPTR EQ SNULL)
                         THEN RADSPTR = TRADS;
                         ELSE;
               ELSE;
     ENDLOOP;

END multi_AC_resolution_logic_advisories_calculations;
```

```
------------------------------------------------------------------------------------------
PROCESS multi_AC_vertical_maneuvered_unmaneuvered_conflict_determination;

    <If an AC that is maneuvered for the current conflict is unmaneuvered in
     another conflict that is being resolved using a vertical advisory, then
     evaluate how the vertical advisory will affect the previous conflict.  If
     the predicted separation for the previous conflict is projected to be
     less than a minimum acceptable separation, do not use this advisory set
     for the current conflict.>

    IF (a maneuvered AC is unmaneuvered in another conflict AND
              resolution for the previous conflict is in the vert dimension)
        THEN save vertical resolution advisory from the RADS for subject AC;
              Save pointer to subject AC and AC in previous conflict with
                   subject AC;
              CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION;
              IF (predicted 3-D separation for previous conflict LT
                             minimum acceptable separation threshold)
                  THEN CLEAR maneuvered_unmaneuvered conflict feature;
                  ELSE;
        ELSE;

END multi_AC_vertical_maneuvered_unmaneuvered_conflict_determination;
```

```
PROCESS multi_AC_vertical_maneuvered_unmaneuvered_conflict_determination;

    FLT PSEP;

    IF (a maneuvered AC is unmaneuvered in another conflict AND resolution for the
            previous conflict is in the vert dimension)
        THEN save vertical RA from the RADS for subject AC;
            Save pointer to subject AC and AC in previous conflict with
                subject AC;
            CALL RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION
                  IN (RAs for both AC, AC state vectors)
                  OUT (PSEP);
            IF (PSEP LT RESADV.SEP1)
                THEN TRADS.FEATBITS(3) = $FALSE;
                ELSE;
        ELSE;

END multi_AC_vertical_maneuvered_unmaneuvered_conflict_determination;
```

```
-------------------------------------------------------------------------------------
PROCESS negative_resolution_advisory_determination;

     IF (either AC is turning)
          THEN add a buffer to the normal horizontal negative resolution advisory
                    threshold;
          ELSE use the normal horizontal negative resolution advisory threshold;
     PERFORM vertical_divergence_logic;
     IF (this advisory has a horizontal component)
          THEN IF (horizontal separation for both AC maneuvering and neither AC
                         maneuvering GT negative horizontal resolution advisory
                         threshold)
                    THEN IF (both AC maneuvered)
                              THEN IF ((horizontal separation for first AC only
                                             maneuvering GT negative separation
                                             threshold) AND horizontal separation for
                                             second AC only maneuvering GT separation
                                             threshold))
                                        THEN SET flag to indicate negative resolution
                                                  advisories are sufficient;
                                        ELSE;
                              ELSE SET flag to indicate negative resolution
                                        advisories are sufficient;
                    ELSE;
     ELSEIF (both aircraft are maneuvered)      <with vertical advisories>
          THEN IF (the vertical separation achieved by modeling negatives GE vertical
                         negative resolution advisory threshold)
                    THEN SET flag indicating negative res adv are sufficient;
                    ELSE;
     OTHERWISE PERFORM one_AC_maneuvering_negative_vertical_resolution_advisory_test;

     IF (negative resolution advisories are sufficient)
          THEN PERFORM positive_to_negative_resolution_advisory_conversion;
          ELSE;

END negative_resolution_advisory_determination;
```

```
--------------------------------------------------------------------------------------------------
PROCESS negative_resolution_advisory_determination;

     IF ((ACID1.TURN EQ $STRNGLFT) OR (ACID1.TURN EQ $STRNGRGT) OR
              (ACID2.TURN EQ $STRNGLFT) OR (ACID2.TURN EQ $STRNGRGT))
          THEN HDTHH = RESADV.HDTHHSQ;
          ELSE HDTHH = RESADV.HDTHSQ;

     PERFORM vertical_divergence_logic;
     IF (RADS.HORIZ EQ $TRUE)
          THEN IF ((HMD2(TRADS.INDEX1,TRADS.INDEX2) GT HDTHH) AND
                         (HMD2(2,2) GT HDTHH))
                    THEN IF (TRADS.CMDED_CMDED EQ $TRUE)
                              THEN IF ((HMD2(TRADS.INDEX1,2) GT HDTHH) AND
                                             (HMD2(2,TRADS.INDEX2) GT HDTHH))
                                        THEN TRADS.NEGATIVE = $TRUE;
                                        ELSE;
                              ELSE TRADS.NEGATIVE = $TRUE;
                    ELSE;
     ELSEIF (TRADS.CMDED_CMDED EQ $TRUE)      <with vertical advisories>
          THEN IF (VMDA($LEV3) GE ASEP**2)
                    THEN TRADS.NEGATIVE = $TRUE;
                    ELSE;
     OTHERWISE PERFORM one_AC_maneuvering_negative_vertical_resolution_advisory_test;

     IF (TRADS.NEGATIVE EQ $TRUE)
          THEN PERFORM positive_to_negative_resolution_advisory_conversion;
          ELSE;

END negative_resolution_advisory_determination;
```

```
--------------------------------------------------------------------------------
PROCESS non_mode_C_resolution_tau_and_proximity_comparisons;

    <The domino object AC does not have mode C data.  Only the horizontal range
     and tau are checked for violating thresholds to determine if a domino
     conflict is predicted.>

    IF ((current range LT immediate range threshold) OR
              (zero LT horizontal tau LT horizontal tau threshold))
        THEN SET domino resolution advisory flag;
        ELSE;

END non_mode_C_resolution_tau_and_proximity_comparisons;
```

```
--------------------------------------------------------------------------------

PROCESS non_mode_C_resolution_tau_and_proximity_comparisons;

     IF ((DRANGE2 LT DRCHD2) OR ((0 LT DTH) AND (DTH LT DTCHDH)))
          THEN DCHDFLG = $TRUE;
          ELSE;

END non_mode_C_resolution_tau_and_proximity_comparisons;
```

```
-------------------------------------------------------------------------------
PROCESS one_AC_maneuvering_negative_vertical_resolution_advisory_test:

    <Determine if a negative vertical resolution advisory is acceptable for
     resolution when only one AC is maneuvered.  If the positive sense of
     the maneuver is away from the unmaneuvered AC, the unmaneuvered AC does
     not have a dangerous vertical velocity and the projected separation for
     the positive sense of the advisory is greater than the vertical
     positive/negative advisory threshold, then the negative sense of the
     vertical advisory is acceptable.>

    IF (resolution advisory is CLIMB AND
              maneuvered AC is lower than unmaneuvered AC)
         THEN:     <"don't descend" won't do>
    ELSEIF (resolution advisory is DESCEND AND
              maneuvered AC is higher than unmaneuvered AC)
         THEN;
    ELSEIF (unmaneuvered AC is converging at a high rate)
         THEN:     <negative dangerous>
    ELSEIF (the vertical miss distance for maneuvered AC modeled as responding
              to the positive vertical resolution advisory LT
              negative vertical resolution advisory separation threshold)
         THEN;
    OTHERWISE SET flag in RADS indicating negative resolution advisories sufficient;

END one_AC_maneuvering_negative_vertical_resolution_advisory_test;
```

```
--------------------------------------------------------------------------------

PROCESS one_AC_maneuvering_negative_vertical_resolution_advisory_test;

<For this process only, ACID1 is the maneuvering AC and
ACID2 is the non-maneuvering AC>

     IF ((TRADS.V EQ $CL) AND (ACID1.Z LT ACID2.Z))
          THEN:      <"don't descend" won't do>
     ELSEIF ((TRADS.V EQ $DES) AND (ACID1.Z GT ACID2.Z))
          THEN:
     ELSEIF (ACID2.ZD GT ZDTH)
          THEN:      <negative dangerous>
     ELSEIF (VMDA(TRADS.INDEX3) LT ASEP**2)
          THEN:
     OTHERWISE TRADS.NEGATIVE = $TRUE;

END one_AC_maneuvering_negative_vertical_resolution_advisory_test;
```

```
------------------------------------------------------------------------------
PROCESS other_sources_resolution_advisory_determination;

    <If the subject AC is maneuvered in the current conflict and is involved
     in more than one conflict pair, then check if the AC is receiving
     resolution advisories from another ATARS site or BCAS.  Save the effective
     horizontal and vertical resolution advisories (if any) from the other
     sources.>

    CLEAR temporary values of resolution advisories from other sources;
    LOOP:
         Get next AC of subject pair;
    EXITIF (done both AC);
         IF ((this AC is receiving resolution advisories from more than one
                   ATARS source or from BCAS) AND
                   (this AC is maneuvered in the current conflict))
              THEN LOOP:
                        Get the next pair record for this conflict table;
                   EXITIF (no more pair records);
                        IF (the resolution advisories in this pair record are from
                                  BCAS OR from a non-connected ATARS site)
                             THEN save the resolution advisories from this source;
                             ELSE;
                   ENDLOOP;
              ELSE;
    ENDLOOP;

END other_sources_resolution_advisory_determination;
```

```
--------------------------------------------------------------------------------
PROCESS other_sources_resolution_advisory_determination;

    OSHMAN1 = $NULLRES;
    OSHMAN2 = $NULLRES;
    OSVMAN1 = $NULLRES;
    OSVMAN2 = $NULLRES;

    LOOP;
        Get next AC of subject pair;
    EXITIF (done both AC);
        IF ((ACID.CTE.NCON GT 1) AND (RSPND EQ $TRUE))
            THEN LOOP;
                    Get next pair record for this conflict table;
                EXITIF (no more pair records);
                    IF ((TPREC.ATSID EQ $BCAS) OR
                            (TPREC.ATSID EQ non-connected site))
                        THEN IF (ACID.CTE EQ TPREC.ac1.PAC)
                                THEN OSHMAN = EFFHRA(OSHMAN,TPREC.ac.PHMAN);
                                     OSVMAN = EFFVRA(OSVMAN,TPREC.ac.PVMAN);
                        ELSE;
                ENDLOOP;
            ELSE;
    ENDLOOP;

END other_sources_resolution_advisory_determination;
```

```
---------------------------------------------------------------------------------
PROCESS pair_record_check_for_existing_potential_domino_conflict_list;

          <Check thru other pair records that the subject AC is in
           to determine if a list of potential domino conflict AC
           has been determined for this AC on this current cycle.>

     LOOP:
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records OR potential domino conflict list already found);
          IF (this pair record not the subject pair record)
              THEN IF (the subject AC is in this pair record AND a list of potential
                             domino conflict AC exists in the pair record)
                        THEN IF (the resolution advisories for the current conflict
                                       are a subset of those used for the potential
                                       domino conflict list created for the other
                                       conflict)
                                  THEN PERFORM potential_domino_conflict_list_copy;
                                  ELSE;
                        ELSE;
              ELSE;
     ENDLOOP;

END pair_record_check_for_existing_potential_domino_conflict_list;
```

AD-A104 148   MITRE CORP MCLEAN VA METREK DIV   F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE   DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-2   FAA-RD-81-45-2   NL

4 OF 7
AD
A104148

```
--------------------------------------------------------------------------------
PROCESS pair_record_check_for_existing_potential_domino_conflict_list;

         <Check thru other pair records that the subject AC is in
          to determine if a list of potential domino conflict AC
          has been determined for this AC on this current cycle.>

    LOOP:
         Get next pair record associated with this conflict table;
    EXITIF (no more pair records OR potential domino conflict list already found);
         IF (TPREC NE PREC)
              THEN IF (((TPREC.ac1.PAC.ACID EQ ACID) OR
                             (TPREC.ac2.PAC.ACID EQ ACID)) AND
                             (pointer to list of potential domino conflict AC
                             for subject AC in this PR is not null))
                        THEN IF (PREC.ACID.CMDFL is a subset of TPREC.ac.CMDFL)
                                 THEN PERFORM potnetial_domino_conflict_list_copy;
                                 ELSE;
                        ELSE;
              ELSE;
    ENDLOOP;

END pair_record_check_for_existing_potential_domino_conflict_list;
```

```
-----------------------------------------------------------------------------

PROCESS positive_to_negative_resolution_advisory_conversion;

     <The 'negative suffices' flag is set for a resolution advisory set.  Convert
      the positive resolution advisories to negative resolution advisories in
      this RADS.  Only single dimension RADS are checked for negative.>

     Convert horizontal or vertical resolution advisories to their negatives;

END positive_to_negative_resolution_advisory_conversion;
```

```
--------------------------------------------------------------------------------

PROCESS positive_to_negative_resolution_advisory_conversion;

    IF (TRADS.HORIZ EQ $TRUE)
        THEN IF (TRADS.H1 EQ $TL)
                 THEN TRADS.H1 = $DTR;
             ELSEIF (TRADS.H1 EQ $TR)
                 THEN TRADS.H1 = $DTL;
             OTHERWISE;

             IF (TRADS.H2 EQ $TL)
                 THEN TRADS.H2 = $DTR;
             ELSEIF (TRADS.H2 EQ $TR)
                 THEN (TRADS.H2 = $DTL;
             OTHERWISE;

        ELSE IF (TRADS.V1 EQ $CL)
                 THEN TRADS.V1 = $DDES;
             ELSEIF (TRADS.V1 EQ $DES)
                 THEN TRADS.V1 = $DCL;
             OTHERWISE;

             IF (TRADS.V2 EQ $CL)
                 THEN TRADS.V2 = $DDES;
             ELSEIF (TRADS.V2 EQ $DES)
                 THEN TRADS.V2 = $DCL;
             OTHERWISE;

END positive_to_negative_resolution_advisory_conversion;
```

```
PROCESS potential_domino_conflict_list_copy;

    <It has been determined that the same list of potential domino conflict AC may
     be used for this subject AC in the current conflict pair as was used for the
     subject AC in another conflict pair processed this scan.  This routine makes a
     copy of a potential domino conflict list, clearing the values of encounter area
     type, multiplicity and the resolution advisory conflict status variables
     in the copied list.>

    LOOP:
         Get next potential domino conflict pointer from previous pair record;
    EXITIF (no more potential domino conflict entries);
         IF (AC in potential domino conflict list entry NE
                   other subject AC of current pair)
              THEN PERFORM potential_domino_conflict_list_entry_addition;
                   Copy state vector pointer from existing PDC List entry to new
                        Potential Domino Conflict List entry;
                   CLEAR encounter area type and multiplicity;
              ELSE;
    ENDLOOP;

END potential_domino_conflict_list_copy;
```

```
--------------------------------------------------------------------------------
PROCESS potential_domino_conflict_list_copy;

     LOOP:
          Get next potential domino conflict pointer from previous pair record;
     EXITIF (no more potential domino conflict entries);
          IF (TPDC_LIST.INTRAC.PAC.ACID NE other AC of current subject pair)
               THEN PERFORM potential_domino_conflict_list_entry_addition;
                    PDC_LIST.INTRAC = TPDC_LIST.INTRAC;
                    PDC_LIST.ENAT = SUNAT;
                    PDC_LIST.MULT = 0;
               ELSE:
     ENDLOOP:

END potential_domino_conflict_list_copy;
```

```
PROCESS potential_domino_conflict_list_creation;

     <This process determines a list of potential domino conflict AC which are
      within the Domino Coarse Screen search limits of the subject AC.>

     IF (this subject AC is in any other conflict pairs)
          THEN PERFORM pair_record_check_for_existing_potential_domino_conflict_list;
          ELSE;

     IF (list of potential domino conflict AC must be determined)
          THEN PERFORM domino_coarse_screen;
          ELSE;          <list of potential domino conflict AC obtained
                         from another pair record>

END potential_domino_conflict_list_creation;
```

```
-------------------------------------------------------------------------------------

PROCESS potential_domino_conflict_list_creation;

    IF (ACID.CTE.NCON GT 1)
        THEN PERFORM pair_record_check_for_existing_potential_domino_conflict_list;
        ELSE:

    IF (PREC.ac.INTR EQ $NULL)
        THEN PERFORM domino_coarse_screen;
        ELSE;          <list of potential domino conflict AC obtained
                       from another pair record>

END potential_domino_conflict_list_creation;
```

```
--------------------------------------------------------------------------------
PROCESS potential_domino_conflict_list_entry_addition;

    <Add an entry to the Potential Domino Conflict List.>

    Link in a new potential domino conflict list entry to the current Potential
        Domino Conflict List;

END potential_domino_conflict_list_entry_addition;
```

```
--------------------------------------------------------------------------------

PROCESS potential_domino_conflict_list_entry_addition;

    Link in a new potential domino conflict list entry to the current Potential
        Domino Conflict List;

END potential_domino_conflict_list_entry_addition;
```

```
PROCESS potential_resolution_advisory_status_variable_determination;

     <This process sets the potential resolution advisory status variables,
      which indicate for which resolution advisories the domino coarse
      screen search must account.>

     CLEAR potential resolution advisory domino status variables;
     LOOP:
          Get next resolution advisory pair from the RADS;
     EXITIF (no potential resolution advisory sets remain);
             ((this potential resolution advisory sets' high-order features value is
                    tied for maximum value) AND
                    (all absolute features are set))
               THEN SET appropriate potential resolution advisory domino status
                         variables;
               ELSE:
     ENDLOOP;

END potential_resolution_advisory_status_variable_determination;
```

```
--------------------------------------------------------------------------------
PROCESS potential_resolution_advisory_status_variable_determination;

     SET potential RA domino status variables to $NOPRA;
     LOOP;
          Get next RA pair from the RADS;
     EXITIF (no potential RA sets remain);
          IF ((this potential RA sets' high-order features value is
                    tied for maximum value) AND (all absolute features are set))
               THEN SET appropriate potential RA domino status variable to $DOMNP;
               ELSE;
     ENDLOOP;

END potential_resolution_advisory_status_variable_determination;
```

```
--------------------------------------------------------------------------------------
PROCESS relative_features_evaluation;

     <These features are called relative, because their function is to order the set
      of potential resolution advisories relative to each other.  It is not necessary
      for any of these features to be set for the potential resolution advisory that
      is ultimately selected to resolve the conflict.>
     <absolute features are first>
     PERFORM feature_PSEP_GE_SEP1;
     IF (RAER not called from Conflict Resolution Data Task)
          THEN PERFORM feature_reinforce_res_adv_from_non_connected_site_or_BCAS;
          ELSE;
     PERFORM feature_terrain_or_obstacle_alert;
     <Domino features are here.>
     PERFORM feature_aircraft_far_from_radar;
     PERFORM feature_negative_resolution_advisories_suffice;
     PERFORM feature_negative_resolution_advisories_do_not_reverse_maneuver;
     PERFORM feature_fast_unmaneuvered_slow_maneuvered;
     PERFORM feature_unmaneuvered_with_large_vertical_rate;
     PERFORM feature_no_level_off_time_for_verticals;
     PERFORM feature_non_response_to_positive_resolution_advisories_detected;
     PERFORM feature_aircraft_on_final_approach;
     PERFORM feature_initial_resolution_advisory_selection;
     PERFORM feature_PSEP_GE_SEP2;
     PERFORM feature_compatible_with_turn;
     PERFORM feature_big_vertical_miss_distance;
     PERFORM feature_big_horizontal_miss_distance;
     PERFORM same_weight_calculations;   <give big_VMD and big_HMD the same weight>
     IF (RAER not called from Conflict Resolution Data Task)
          THEN PERFORM feature_reinforces_prior_resolution_advisories;
          ELSE;
     PERFORM feature_speed_check;
     PERFORM feature_reinforces_turn;
     <tie-breaking feature is last>

END relative_features_evaluation;
-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  ------------
```

```
---------------------------------------------------------------------------
PROCESS relative_features_evaluation;

     <These features are called relative, because their function is to order the set
      of potential RAS relative to each other.  It is not necessary for any of these
      features to be set for the potential RA that is selected for resolution.>

     <absolute features are first>
     PERFORM feature_PSEP_GE_SEP1;
     IF (HRNCAP EQ $TRUE)
          THEN PERFORM feature_reinforce_res_adv_from_non_connected_site_or_BCAS;
          ELSE;
     PERFORM feature_terrain_or_obstacle_alert;
     <Domino features are here.>
     PERFORM feature_aircraft_far_from_radar;
     PERFORM feature_negative_resolution_advisories_suffice;
     PERFORM feature_negative_resolution_advisories_do_not_reverse_maneuver;
     PERFORM feature_fast_unmaneuvered_slow_maneuvered;
     PERFORM feature_unmaneuvered_with_large_vertical_rate;
     PERFORM feature_no_level_off_time_for_verticals;
     PERFORM feature_non_response_to_positive_resolution_advisories_detected;
     PERFORM feature_aircraft_on_final_appreach;
     PERFORM feature_initial_resolution_advisory_selection;
     PERFORM feature_PSEP_GE_SEP2;
     PERFORM feature_compatible_with_turn;
     PERFORM feature_big_vertical_miss_distance;
     PERFORM feature_big_horizontal_miss_distance;
     PERFORM same_weight_calculations;
     IF (HRNCAP EQ $TRUE)
          THEN PERFORM feature_reinforces_prior_resolution_advisories;
          ELSE;
     PERFORM feature_speed_check;
     PERFORM feature_reinforces_turn;
     <tie-breaking feature is last>

END relative_features_evaluation;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_compatibility_with_existing_conflicts;

     <If no resolution advisory sets have all absolute features set after
      evaluating the two-AC definition of the features, evaluate the multi-AC
      definition of the maneuvered_unmaneuvered conflict feature.>

     LOOP;
          Get next potential resolution advisory sets;
     EXITIF (no more potential resolution advisories sets);
          IF (all absolute features other than maneuvered_unmaneuvered are set)
               THEN PERFORM feature_maneuvered_unmaneuvered_conflict_multi_AC_
                                                                    definition;
               ELSE;
          IF (all absolute features set)
               THEN increment counter of RADS with all absolute features set;
               ELSE;
     ENDLOOP;

END resolution_advisory_compatibility_with_existing_conflicts;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_compatibility_with_existing_conflicts;

    LOOP;
        Get next potential RA sets;
    EXITIF (no more potential RAS sets);
        IF ((TRADS.FEATBITS(1) EQ $TRUE) AND (TRADS.FEATBITS(2) EQ $TRUE))
                AND (TRADS.FEATBITS(3) EQ $FALSE))
            THEN PERFORM feature_maneuvered_unmaneuvered_conflict_multi_AC_
                                                                definition;
            ELSE;
        IF ((TRADS.FEATBITS(1) EQ $TRUE) AND (TRADS.FEATBITS(2) EQ $TRUE)
                AND (TRADS.FEATBITS(3) EQ $TRUE))
            THEN NPRAABS = NPRAABS + 1;
            ELSE;
    ENDLOOP;

END resolution_advisory_compatibility_with_existing_conflicts;
```

```
---------------------------------------------------------------------------------------

PROCESS same_weight_calculations;

     <The effect of this process is to give 'large horizontal miss distance' and
      'large vertical miss distance' the same weight.>

     LOOP:
          Get the next RADS;
     EXITIF (no more RADS);
          IF (large vertical miss distance feature set)
               THEN IF (large horizontal miss distance feature not set)
                         THEN SET large horizontal miss distance feature;
                              Add this feature's weight to the RADS total value;
                         ELSE;
               ELSE IF (large horizontal miss distance feature is set)
                         THEN SET large vertical miss distance feature;
                              Add this feature's weight to this RADS total value;
                         ELSE;
     ENDLOOP;

END same_weight_calculations;
```

```
--------------------------------------------------------------------------------
PROCESS same_weight_calculations;

     LOOP:
          Get the next RADS;
     EXITIF (no more RADS);
          IF (TRADS.FEATBITS(20) EQ $TRUE)
               THEN IF (TRADS.FEATBITS(21) = $FALSE;
                         THEN TRADS.FEATBITS(21) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + BIGHWGT;
                         ELSE;
               ELSE IF (TRADS.FEATBITS(21) EQ $TRUE)
                         THEN TRADS.FEATBITS(20) = $TRUE;
                              TRADS.VALUE = TRADS.VALUE + BIGVWGT;
                         ELSE;

END same_weight_calculations;
```

```
PROCESS tie_breaker_features_evaluation;

     <This process is performed whenever two or more potential resolution advisory
      sets are found to be equal in value after the relative features have been
      evaluated.  This process compares predicted separation values to select a
      single resolution advisory set as the best.>

     IF (count of maximum-value resolution advisories GT 1)

          THEN IF (negative suffices for maximum-value resolution advisories)
                    THEN PERFORM feature_biggest_separation_for_negatives;
                    ELSE PERFORM feature_biggest_separation_for_positives;

          ELSE:   < no tie >

END tie_breaker_features_evaluation;
```

```
----------------------------------------------------------------------------------

PROCESS tie_breaker_features_evaluation;

    IF (NUMPRA GT 1)

        THEN IF (RADSPTR.NEGATIVE EQ $TRUE)
                THEN PERFORM feature_biggest_separation_for_negatives;
                ELSE PERFORM feature_biggest_separation_for_positives;

        ELSE;   < no tie >

END tie_breaker_features_evaluation;
```

```
PROCESS two_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;

    <If a maneuvered AC in the subject pair is unmaneuvered in another pair
     that is being resolved using a horizontal resolution advisory, then
     this RADS may not be used to resolve this conflict.>

    LOOP;
        Get next pair record associated with this conflict table;
    EXITIF (no more pair records);
        IF (subject AC in this pair record AND this is not the subject pair record)
            THEN IF (subject AC has no horizontal resolution advisory and other
                           AC has a horizontal resolution advisory)
                    THEN CLEAR maneuvered_unmaneuvered conflict feature;
                    ELSE;
            ELSE;
    ENDLOOP;

END two_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;
```

```
PROCESS two_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;

    LOOP:
        Get next pair record associated with this conflict table;
    EXITIF (no more pair records);
        IF ((ACID EQ TPREC.ac1.PAC.ACID) OR (ACID EQ TPREC.ac2.PAC.ACID)
                AND (TPREC NE PREC))
            THEN IF (TPREC.PAC.MMAN EQ $NONES)
                THEN TRADS.FEATBITS(3) = $FALSE;
                ELSE;
            ELSE;
    ENDLOOP;

END two_AC_horizontal_maneuvered_unmaneuvered_conflict_determination;
```

```
--------------------------------------------------------------------------------
PROCESS two_AC_resolution_logic_vertical_resolution_advisories_selection;

    <This is the 'eight second rule' described in the text.  This logic
     selects opposite sense vertical resolution advisories for each AC.>

    IF (RAER called from Conflict Resolution Data Task)
        <a pair record may not exist if called from Conflict Resolution Data Task>
        THEN project altitude of each AC ahead eight (8) seconds;
            IF (projected altitude of first AC GE projected altitude of second AC)
                THEN vertical resolution advisory for first AC is Climb;
                     Vertical resolution advisory for second AC is Descend;
                ELSE vertical resolution advisory for second AC is Climb;
                     Vertical resolution advisory for first AC is Descend;

        ELSE IF (a vertical resolution advisory is in the pair record)
                THEN save the vertical resolution advisories in the pair record
                          as the selected vertical resolution advisories;
                     IF (either AC does not have a vert res adv)
                          THEN SET vert res adv for that AC to vert res adv
                                    opposite to that of other AC;
                          ELSE;
                ELSE project altitude of each AC ahead eight seconds;
                     IF (altitude of first AC GE
                               projected altitude of second AC)
                          THEN vertical resolution advisory for first AC is
                                    climb;
                               Vertical resolution advisory for second AC is
                                    descend;
                          ELSE vertical resolution advisory for first AC is
                                    descend;
                               Vertical resolution advisory for second AC is
                                    climb;

END two_AC_resolution_logic_vertical_resolution_advisories_selection;

-----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  -------------
```

```
--------------------------------------------------------------------------------
PROCESS two_AC_resolution_logic_vertical_resolution_advisories_selection;

    IF (MRNCAP EQ $FALSE)
        THEN Z8SEC1 = ACID1.Z + ACID1.ZD * TVRULE;
             Z8SEC2 = ACID2.Z + ACID2.ZD * TVRULE;
             IF (Z8SEC1 GE Z8SEC2)
                 THEN VERTRA1 = $CL;
                      VERTRA2 = $DES;
                 ELSE VERTRA2 = $CL;
                      VERTRA1 = $DES;

        ELSE IF ((PREC.ac1.PVMAN EQ $NULLRES) OR (PREC.ac1.PVMAN EQ $NORES) OR
                     (PREC.ac2.PVMAN EQ $NULLRES) OR (PREC.ac2.PVMAN EQ $NORES))
                 THEN VERTRA1 = PREC.ac1.PVMAN;
                      VERTRA2 = PREC.ac2.PVMAN;
                      IF (VERTRA1 EQ $NORES)
                          THEN VERTRA1 = opposite vertical RA to VERTRA2;
                          ELSE IF (VERTRA2 EQ $NORES)
                                   THEN VERTRA2 = opposite vertical RA to
                                            VERTRA1;
                                   ELSE;
                 ELSE Z8SEC1 = ACID1.Z + ACID1.ZD * TVRULE;
                      Z8SEC2 = ACID2.Z + ACID2.ZD * TVRULE;
                      IF (Z8SEC1 GE Z8SEC2)
                          THEN VERTRA1 = $CL;
                               VERTRA2 = $DES;
                          ELSE VERTRA1 = $DES;
                               VERTRA2 = $CL;

END two_AC_resolution_logic_vertical_resolution_advisories_selection;
```

```
---------------------------------------------------------------------------
PROCESS two_AC_vertical_maneuvered_unmaneuvered_conflict_determination;

     <If a maneuvered AC in the subject pair is unmaneuvered in another pair
      that is being resolved using a vertical resolution advisory, then
      this RADS may not be used to resolve this conflict.>

     LOOP:
          Get next pair record associated with this conflict table;
     EXITIF (no more pair records):
          IF (subject AC in this pair record AND this is not the subject pair record)
               THEN IF (subject AC has no vertical resolution advisory and other AC
                              has a vertical resolution advisory)
                         THEN CLEAR maneuvered_unmaneuvered conflict feature;
                         ELSE;
               ELSE;
     ENDLOOP;

END two_AC_vertical_maneuvered_unmaneuvered_conflict_determination;
```

```
--------------------------------------------------------------------------------

PROCESS two_AC_vertical_maneuvered_unmaneuvered_conflict_determination;

    LOOP:
        Get next pair record associated with this conflict table;
    EXITIF (no more pair records);
        IF ((ACID EQ TPREC.ac1.PAC.ACID) OR (ACID EQ TPREC.ac2.PAC.ACID)
                AND (TPREC NE PREC))
            THEN IF (TPREC.VMAN EQ $NOMES)
                    THEN TRADS.FEATBITS(3) = $FALSE;
                    ELSE;
            ELSE;
    ENDLOOP;

END two_AC_vertical_maneuvered_unmaneuvered_conflict_determination;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_divergence_logic;

     Compute true horizontal tau;
     IF (AC are converging vertically)
          THEN:
     ELSEIF (relative altitude difference LT
               negative vertical resolution advisory threshold)
          THEN:
     ELSEIF (AC are diverging horizontally)
          THEN:
     OTHERWISE determine the look-ahead time as the minimum of true tau, and a
               parameter;
          Compute relative altitude separation at the look-ahead time;
          IF (relative altitude separation GT
                    negative vertical resolution advisory threshold)
               THEN SET values in VMDA array for AC maneuvering vertically to
                         relative altitude separation at look-ahead time;
               ELSE:

END vertical_divergence_logic;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_divergence_logic;

     FLT (TM, TZ1, TZ2, TVMD);

     TRTHU = ELENTRY.RANGE2 / ELENTRY.DOT;
     IF (ELENTRY.TV GE 0)
          THEN;
     ELSEIF (ELENTRY.ALT LE NSVDAT)
          THEN;
     ELSEIF (TRTHU LE 0)
          THEN;
     OTHERWISE TM = MIN(TRTHU,NSVDTT);
               TZ1 = ACID1.Z + (ACID1.ZD * TM);
               TZ2 = ACID2.Z + (ACID2.ZD * TM);
               TVMD = ABS(TZ2 - TZ1);
               IF (TVMD GT ASEP)
                    THEN VMDA(2) = TVMD**2;
                         VMDA(3) = TVMD**2;

END vertical_divergence_logic;
```

```
--------------------------------------------------------------------------------

PROCESS X_list_backward_domino_search;

    <Search backwards (decreasing X values) on the X-list until the lower domino
     search limit is reached or there are no more AC.  Do not include state vectors
     that are signposts or AC that are currently in conflict with the subject AC.
     Also, don't include AC in a final approach zone if the subject AC is also
     in a final approach zone.>

    LOOP;
        Get next AC in direction of decreasing X on X-list;
    EXITIF (no more AC OR X position of next AC LT X lower limit);
        IF ((next AC not in a conflict pair with the subject AC) AND
                  (next state vector is not a signpost) AND
                  (both AC are not in a final approach zone))
             THEN IF (next AC Y position within Y search limits)
                       THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                       ELSE;
             ELSE;
    ENDLOOP

END X_list_backward_domino_search;
```

```
-----------------------------------------------------------------------------------

PROCESS X_list_backward_domino_search;

     LOOP:
          Get next AC in direction of decreasing X on X-list;
     EXITIF (no more AC OR X position of next AC LT X lower limit);
          IF ((next AC not in a conflict pair with the subject AC) AND
                    (NXTAC.SPIDFG EQ $FALSE) AND
                    (both AC are not in a final approach zone))
               THEN IF ((YL LE NXTAC.Y) AND (NXTAC.Y LE YU))
                         THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                         ELSE;
               ELSE;
     ENDLOOP

END X_list_backward_domino_search;
```

```
-----------------------------------------------------------------------------

PROCESS X_list_domino_search;

     <This procedure performs the search of the X-list around the
      subject AC within the domino coarse screen search limits.>

     PERFORM X_list_forward_domino_search;

     PERFORM X_list_backward_domino_search;

END X_list_domino_search;
```

```
--------------------------------------------------------------------------------

PROCESS X_list_domino_search;

     PERFORM X_list_forward_domino_search;

     PERFORM X_list_backward_domino_search;

END X_list_domino_search;
```

```
--------------------------------------------------------------------------------

PROCESS X_list_domino_search_limits_calculations;

     <Calculate the X-list search limits by adding the X-list domino buffer area
      to the subject AC domino area.>

     Add maximum horizontal range to upper X & Y values of subject AC domino area;

     Subtract maximum horizontal range from lower X & Y values of subject AC
      domino area;

     Add maximum vertical range to upper Z value of subject AC domino area;

     Subtract maximum vertical range from lower Z value of subject AC domino area;

END X_list_domino_search_limits_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS X_list_domino_search_limits_calculations;

     XU = XMAX + RMAX;
     YU = YMAX + RMAX;
     ZU = ZMAX + ZMX;

     XL = XMIN - RMAX;
     YL = YMIN - RMAX;
     ZL = ZMIN - ZMX;

END X_list_domino_search_limits_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS X_list_forward_domino_search;

    <Search forward (increasing X values) on the X-list until the upper domino
     search limit is reached or there are no more AC.  Do not include state vectors
     that are signposts or AC that are currently in conflict with the subject AC.
     Also, don't include AC in a final approach zone if the subject AC is also
     in a final approach zone.>

    LOOP:
        Get next AC in direction of increasing X on X-list;
    EXITIF (no more AC OR X position of next AC GT upper X limit):
        IF ( AC not in a conflict pair with the subject AC) AND (next
                  (next state vector is not a signpost) AND
                  (both AC are not in a final approach zone))
             THEN IF (next AC Y position within Y search limits)
                       THEN PERFORM domino_coarse_screen_altitude_conflict_test:
                       ELSE;
             ELSE;
    ENDLOOP;

END X_list_forward_domino_search;
```

```
---------------------------------------------------------------------------------------

PROCESS X_list_forward_domino_search;

     LOOP;
          Get next AC in direction of increasing X on X-list;
     EXITIF (no more AC OR NXTAC.X GT XU);
          IF ((next AC not in a conflict pair with the subject AC) AND
                    (NXTAC.SPIDFG EQ $FALSE) AND
                    (both AC are not in a final approach zone))
               THEN IF ((YL LE NXTAC.Y) AND (NXTAC.Y LE YU))
                         THEN PERFORM domino_coarse_screen_altitude_conflict_test;
                         ELSE;
               ELSE;
     ENDLOOP;

END X_list_forward_domino_search;
```

```
--------------------------------------------------------------------------------------

PROCESS X_list_object_AC_domino_buffer_area_calculations;

     <Calculate the max distance that an AC on the X-list can
      travel during the domino projection interval.  This distance
      is based on the max speed of an AC on the X-list, an assumed
      vertical velocity and the maximum detection threshold values.>

     Calculate the maximum horizontal rangeas: maximum X-list velocity *
          (modeling delay period + 4 * scan time + maximum detect threshold) +
          max immediate range separation threshold;

     Calculate the maximum vertical range as: maximum vertical velocity *
          (modeling delay period + 4 * scan time + maximum detection threshold);

END X_list_object_AC_domino_buffer_area_calculations;
```

```
------------------------------------------------------------------------------------

PROCESS X_list_object_AC_domino_buffer_area_calculations;

     <Calculate the max distance that an AC on the X-list can
      travel during the domino projection interval.  This distance
      is based on the max speed of an AC on the X-list, an assumed
      vertical velocity and the maximum detection threshold values.>

     RMAX = XVEL * (DELAY + DOMSCANS * SYSTEM.SCANT + TLD) + PDVBL.RCONTH(3);

     ZMX = CSCREEN.ZFAST * (DELAY + DOMSCANS * SYSTEM.SCANT + TLD);

END X_list_object_AC_domino_buffer_area_calculations;
```

```
--------------------------------------------------------------------------------

ROUTINE COMPUTATION_OF_TURN_CONSTANTS
   IN (Horizontal velocity of aircraft, time interval)
   OUT (Turn constants);

     < This routine computes the constants used to model a turn. >

     Compute turn rate in radians/sec, assuming a bank angle of BANKA;

     Compute turn constants from turn rate and time interval;

END COMPUTATION_OF_TURN_CONSTANTS;
```

```
--------------------------------------------------------------------------------

ROUTINE COMPUTATION_OF_TURN_CONSTANTS
   IN (VSQ, TINT)
   OUT (GROUP TURCON.ac)

     FLT W;

     W = G * TAN(BANKA) / SQRT(VSQ);

     SA = SIN(W * TINT);
     CA = COS(W * TINT);
     A = (1 - CA)/W;
     B = SA / W;

END COMPUTATION_OF_TURN_CONSTANTS;
```

```
--------------------------------------------------------------------------------
ROUTINE CONTINUE_STRAIGHT
   IN (Projection time)
   INOUT (X,Y components of position and velocity);

     < This routine projects an aircraft straight ahead horizontally. >

     Compute new X,Y positional coordinates for the specified
        velocity and projection time interval;

END CONTINUE_STRAIGHT;
```

```
--------------------------------------------------------------------------------
ROUTINE CONTINUE_STRAIGHT
   IN (TINT)
   INOUT (GROUP GEOM.hor);

     GEOM.X = GEOM.X + GEOM.XD * TINT;
     GEOM.Y = GEOM.Y + GEOM.YD * TINT;

END CONTINUE_STRAIGHT;
```

```
--------------------------------------------------------------------------------------

ROUTINE CONVERGENCE_HORIZONTAL
   IN (Horizontal positions and velocities of two aircraft)
   OUT (Horizontal relative positions and velocities,
        indication of horizontal convergence or divergence);

     < This routine determines horizontal convergence. >

     Compute horizontal position of aircraft 2 relative to aircraft 1;

     Compute horizontal velocity of aircraft 2 relative to aircraft 1;

     Compute horizontal DOT:   < range * range rate >

END CONVERGENCE_HORIZONTAL;
```

```
--------------------------------------------------------------------------------

ROUTINE CONVERGENCE_HORIZONTAL
   IN (GROUP GEOM.hor1, GROUP GEOM.hor2)
   OUT (GROUP MODVBL.relative_geometry);

     RX = GEOM.hor2.X - GEOM.hor1.X;
     RY = GEOM.hor2.Y - GEOM.hor1.Y;

     VRX = GEOM.hor2.XD - GEOM.hor1.XD;
     VRY = GEOM.hor2.YD - GEOM.hor1.YD;
     VR2 = VRX**2 + VRY**2;

     DOT = VRX * RX + VRY * RY;

 END CONVERGENCE_HORIZONTAL;
```

```
--------------------------------------------------------------------------------
ROUTINE CONVERGENCE_3D
   IN (Positions and velocities of two aircraft)
   OUT (Relative positions and velocities (vertical weighted),
        indication of convergence or divergence);

     < This routine determines 3-D convergence. Vertical is weighted by VWEIGHT:1. >

     Compute position of aircraft 2 relative to aircraft 1;

     Compute velocity of aircraft 2 relative to aircraft 1;

     Compute 3-D DOT;   < slant range * rate-of-change of slant range >

END CONVERGENCE_3D;
```

```
--------------------------------------------------------------------------------

ROUTINE CONVERGENCE_3D
   IN (GROUP GEOM.hor1, GROUP GEOM.ver1, GROUP GEOM.hor2, GROUP GEOM.ver2)
   OUT (GROUP MODVBL.relative_geometry);

      RX = GEOM.hor2.X - GEOM.hor1.X;
      RY = GEOM.hor2.Y - GEOM.hor1.Y;
      RZ = (GEOM.ver2.Z - GEOM.ver1.Z) * VWEIGHT;

      VRX = GEOM.hor2.XD - GEOM.hor1.XD;
      VRY = GEOM.hor2.YD - GEOM.hor1.YD;
      VRZ = (GEOM.ver2.ZD - GEOM.ver1.ZD) * VWEIGHT;
      VR2 = VRX**2 + VRY**2 + VRZ**2;

      DOT = RX * VRX + RY * VRY + RZ * VRZ;

END CONVERGENCE_3D;
```

```
--------------------------------------------------------------------------------
ROUTINE DOMINO_RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
   IN  (resolution advisory thresholds)
   OUT (resolution advisory flag);

     <Determines if resolution advisory flag is set for the subject AC-domino
      object AC pair.>

     IF (AC violating horizontal resolution envelope)
          THEN indicate horizontal proximity;
     ELSEIF (AC will violate horizontal resolution envolope soon)
          THEN:
     OTHERWISE failed horizontal tests;

     IF (AC violating vertical resolution envelope presently)
          THEN indicate vertical proximity;
     ELSEIF (AC will be coaltitude soon)
          THEN:
     OTHERWISE failed vertical tests;

     IF (all tests passed)
          THEN SET output flag;
          ELSE;

END DOMINO_RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
```

```
--------------------------------------------------------------------------------

ROUTINE DOMINO_RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
   IN (STRUCTURE DRAVBL)
   OUT (DCMDFLG);

     FLAG NORES;

     NORES = $FALSE;

     IF (DRANGE2 LT DRCMD2 * RAPARM.BZP2)
          THEN:
     ELSEIF (DTH LT DTCMDH)
          THEN:
     OTHERWISE NORES = $TRUE;

     IF (DALT LT DAF * RAPARM.BZP)
          THEN:
     ELSEIF ((DTV GE 0) AND (DTV LE DTCMDV))
          THEN:
     OTHERWISE NORES = $TRUE;

     IF (NORES EQ $FALSE)
          THEN DCMDFLG = $TRUE;
          ELSE DCMDFLG = $FALSE;

END DOMINO_RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
```

```
ROUTINE DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
   IN  (absolute value of relative vertical velocity, encounter area type,
        conflict multiplicity, pair equipment and control status,
        and convergence/divergence rate)
   OUT (resolution advisory detection thresholds);

     IF (there is a controlled AC in the pair)
        THEN calculate the horizontal controller alert tau threshold;

             Calculate the vertical controller alert tau threshold;
             Set resolution advisory thresholds based on controller alert thresholds;

        ELSE CALL DOMINO_UNCON_UNCON_INDEX_DETERMINATION;
             Set resolution advisory thresholds based on number of AC in the conflict;

END DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;
```

```
--------------------------------------------------------------------------------------------

ROUTINE DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
   IN  (VRZA,ENAT,MULT,PREQ,PRCONT,DOT)
   OUT (GROUP DRAVBL);

     FLT (VRZA, DOT);
     INT (ENAT, MULT, PREQ, PRCONT);

     IF (PRCONT NE $NOCONT)
          THEN TCONH = PDVBL.TWARN - ((PDVBL.RCONTH * R) / DOT;
               VRZA = MIN(VRZA, THRSPARM.VRZCON);
               TCONV = PDVBL.TWARN - (PDVBL.ACONTH / VRZA);
               SET GROUP DRAVBL.thresholds as defined in Table 13-11;

          ELSE CALL DOMINO_UNCON_UNCON_INDEX_DETERMINATION
                        IN  (MULT)
                        OUT (UUIND);
               SET GROUP DRAVBL.thresholds using Table 13-11;

END DOMINO_TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;
```

```
--------------------------------------------------------------------------------
ROUTINE DOMINO_UNCON_UNCON_INDEX_DETERMINATION
   IN  (MULT)
   OUT (UUIND);

     IF (number of AC in this conflict cluster GE 4)
          THEN SET index to 2;
     ELSEIF (neither or both ATARS-equipped)
          THEN SET index to 1;
     OTHERWISE IF (ratio of equipped AC's speed to unequipped AC's speed LT
                     threshold)
                    THEN SET index to 2;
                    ELSE SET index to 1;

END DOMINO_UNCON_UNCON_INDEX_DETERMINATION;
```

```
--------------------------------------------------------------------------------
ROUTINE DOMINO_UNCON_UNCON_INDEX_DETERMINATION
   IN  (MULT)
   OUT (UUIND);

     FLT VRAT;
     INT (MULT, UUIND);
     INT MULTIAC;                 <number of AC in multiple AC conflict (4)>
     INT TWO;                     <local constant: 2>

     IF (MULT GE MULTIAC)
          THEN UUIND = TWO;
     ELSEIF (neither or both ATARS-equipped)
          THEN UUIND = 1;
     OTHERWISE VRAT = VSQ(equipped_AC) / VSQ(unequipped_AC);
                IF (VRAT LT PDPARM.VRATTH);
                     THEN UUIND = TWO;
                     ELSE UUIND = 1;

END DOMINO_UNCON_UNCON_INDEX_DETERMINATION;
```

```
--------------------------------------------------------------------------------
ROUTINE FINAL_VERTICAL_RATE_DETERMINATION
   IN (Current vertical rate and horizontal velocity for an aircraft,
       vertical resolution advisory to be modeled)
   OUT (Final vertical rate to be achieved);

     < This routine determines the final vertical rate to be modeled for
       an aircraft for a specified vertical resolution advisory. >

     IF (resolution advisory EQ 'climb')
          THEN IF (this is a 'fast' aircraft)
                    THEN minimum vertical rate = ZDUPF;
                    ELSE minimum vertical rate = ZDUPS;
               Maximum vertical rate = large positive value;

     ELSEIF (resolution advisory EQ 'descend')
          THEN IF (this is a 'fast' aircraft)
                    THEN maximum vertical rate = -ZDDWNF;
                    ELSE maximum vertical rate = -ZDDWNS;
               Minimum vertical rate = large negative value;

     OTHERWISE IF (resolution advisory contains 'don't climb' or 'limit climb')
                    THEN select maximum vertical rate from ZDMAX table;
                    ELSE maximum vertical rate = large positive value;
               IF (resolution advisory contains 'don't descend' or 'limit descent')
                    THEN select minimum vertical rate from ZDMIN table;
                    ELSE minimum vertical rate = large negative value;

     IF (current vertical rate LT minimum vertical rate)
          THEN final vertical rate = minimum vertical rate;
     ELSEIF (current vertical rate GT maximum vertical rate)
          THEN final vertical rate = maximum vertical rate;
     OTHERWISE final vertical rate = current vertical rate;

END FINAL_VERTICAL_RATE_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE FINAL_VERTICAL_RATE_DETERMINATION
   IN (ZD, VSQ, VERTRA)
   OUT (ZDF);

     FLT (ZDMIN, ZDMAX);

     IF (VERTRA EQ $CL)
          THEN IF (VSQ GT VTHSQ)
                    THEN ZDMIN = ZDUPF;
                    ELSE ZDMIN = ZDUPS;
               ZDMAX = large positive value;

     ELSEIF (VERTRA EQ $DES)
          THEN IF (VSQ GT VTHSQ)
                    THEN ZDMAX = -ZDDWNF;
                    ELSE ZDMAX = -ZDDWNS;
               ZDMIN = large negative value;

     OTHERWISE IF (VERTRA contains 'don't climb' or 'limit climb')
                    THEN select ZDMAX from ZDMAX table;   < Table 13-6 >
                    ELSE ZDMAX = large positive value;
               IF (VERTRA contains 'don't descend' or 'limit descent')
                    THEN select ZDMIN from ZDMIN table;   < Table 13-6 >
                    ELSE ZDMIN = large negative value;

     IF (ZD LT ZDMIN)
          THEN ZDF = ZDMIN;
     ELSEIF (ZD GT ZDMAX)
          THEN ZDF = ZDMAX;
     OTHERWISE ZDF = ZD;

END FINAL_VERTICAL_RATE_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE MISS_DISTANCE_HORIZONTAL
   IN (Horizontal relative positions and velocities of two aircraft)
   OUT (Horizontal miss distance);

     < This routine computes horizontal miss distance, assuming straight flight. >

     IF (magnitude of relative horizontal velocity is very small)
          THEN horizontal miss distance = current range;
          ELSE compute horizontal miss distance from relative position and velocity;

END MISS_DISTANCE_HORIZONTAL;
```

```
--------------------------------------------------------------------------------

ROUTINE MISS_DISTANCE_HORIZONTAL
   IN (GROUP MODVBL.relative_geometry)
   OUT (MD2);

     IF (VR2 LT VRTH2)
          THEN MD2 = RX**2 + RY**2;
          ELSE MD2 = (RX * VRY  - RY * VRX)**2 / VR2;

END MISS_DISTANCE_HORIZONTAL;
```

```
ROUTINE MISS_DISTANCE_3D
   IN (Relative positions and velocities of two aircraft,
       indication of whether vertical component is to be calculated)
   OUT (3-D miss distance (vertical weighted),
        unweighted vertical component of 3-D miss distance);

     < This routine computes 3-D miss distance, assuming straight flight. >

     IF (magnitude of relative velocity is very small)

          THEN 3-D miss distance = current slant range;
               IF (vertical component is to be calculated)
                    THEN vertical component = current vertical separation;

          ELSE Compute 3-D miss distance from relative position and velocity;
               IF (vertical component is to be calculated)
                    THEN Compute unweighted vertical component of 3-D miss distance;

END MISS_DISTANCE_3D;
```

```
--------------------------------------------------------------------------------

ROUTINE MISS_DISTANCE_3D
   IN (GROUP MODVBL.relative_geometry, VERTCOMP)
   OUT (MD2, VCMD);

     FLT (A, B, C, TOCA);

     IF (VR2 LT VRTH2)

          THEN MD2 = RX**2 + RY**2 + RZ**2;
               IF (VERTCOMP EQ $TRUE)
                    THEN VCMD = ABS(RZ) / VWEIGHT;
                    ELSE VCMD = 0;

          ELSE A = RY * VRZ - RZ * VRY;
               B = RZ * VRX - RX * VRZ;
               C = RX * VRY - RY * VRX;
               MD2 = (A**2 + B**2 + C**2) / VR2;

               IF (VERTCOMP EQ $TRUE)
                    THEN TOCA = -DOT / VR2;
                         VCMD = ABS(RZ + VRZ * TOCA) / VWEIGHT;
                    ELSE VCMD = 0;

END MISS_DISTANCE_3D;
```

```
-------------------------------------------------------------------------------
ROUTINE NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING
   IN (Pointer to aircraft state vector, vertical resolution advisory to be modeled)
   INOUT (RAPP table):

     < This routine models a negative vertical resolution advisory for one aircraft,
       and stores the resulting projections in the RAPP table. >

     Access aircraft state vector;
     Access conflict table entry via pointer in state vector;
     Obtain previous vertical RA (if any) from conflict table entry;
     Initialize altitude and vertical rate to current values;

     < Model the delay period. >

     IF (aircraft has a previous vertical resolution advisory)
          THEN < aircraft is in linear vertical flight. >
               CALL VERTICAL_ADVANCEMENT;   < Use current vertical rate. >
          ELSE < aircraft may be in nonlinear vertical flight. >
               PERFORM vertical_only_nonlinear_modeling_of_delay;

     < Model the maneuver period. >

     PERFORM vertical_only_modeling_of_maneuver_period;

END NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING;
```

```
--------------------------------------------------------------------------------------

ROUTINE NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING
   IN (ACID, VERTRA)
   INOUT (RAPP table);

     Access SVECT via ACID;

     IF (SVECT.CTE NE $NULL)
          THEN VRAP = SVECT.CTE->CTENTRY.VMAND;
          ELSE VRAP = $NORES;

     Z = SVECT.Z;
     ZD = SVECT.ZD;

     IF (VRAP NE $NORES)

          THEN CALL VERTICAL_ADVANCMENT
                       IN (ZD, DELINT)
                       INOUT (NVGEOM.ver);

          ELSE PERFORM vertical_only_nonlinear_modeling_of_delay;

     PERFORM vertical_only_modeling_of_maneuver_period;

END NEGATIVE_VERTICAL_RESOLUTION_ADVISORY_MODELING;
```

```
--------------------------------------------------------------------------------------
PROCESS vertical_only_nonlinear_modeling_of_delay;

     < This process models the vertical profile of an aircraft during the delay
       period when a previous vertical resolution advisory is being displayed. >

     FLT TIME;   < local variable >

     CALL FINAL_VERTICAL_RATE_DETERMINATION;   < Use previous displayed vertical RA >

     TIME = 0;

     REPEAT UNTIL (TIME GE DELAY);

          < Advance aircraft by DELINT seconds. >

          IF (last half of delay period)
               THEN < respond to any previous vertical advisories. >
                    CALL VERTICAL_ADVANCEMENT;
               ELSE < advance at current vertical rate. >
                    CALL VERTICAL_ADVANCEMENT;

          TIME = TIME + DELINT;

     ENDREPEAT;

END vertical_only_nonlinear_modeling_of_delay;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_only_nonlinear_modeling_of_delay;

     PLT (TIME, ZDF);

     CALL FINAL_VERTICAL_RATE_DETERMINATION
             IN (ZD, SVECT.VSQ, VRAP)
             OUT (ZDF);

     TIME = 0;

     REPEAT UNTIL (TIME GE DELAY);

          IF (TIME GE DELAY/2)
               THEN CALL VERTICAL_ADVANCEMENT
                             IN (ZDF, DELINT)
                             INOUT (NVGEOM.ver);
               ELSE CALL VERTICAL_ADVANCEMENT
                             IN (ZD, DELINT)
                             INOUT (NVGEOM.ver);

          TIME = TIME + DELINT;

     ENDREPEAT;

END vertical_only_nonlinear_modeling_of_delay;
```

```
-------------------------------------------------------------------------------

PROCESS vertical_only_modeling_of_maneuver_period;

    < This process models the vertical profile of an aircraft responding to a
      specified vertical resolution advisory and stores the results in the RAPP
      table for the aircraft. >

    FLT TIME;   < local variable >

    CALL FINAL_VERTICAL_RATE_DETERMINATION;   < Use vertical RA to be modeled. >

    TIME = TIMINT / 2;   < Use time at middle of each interval. >

    REPEAT WHILE (TIME LE TNVRAM);

        < Advance aircraft by TIMINT seconds. >

        CALL VERTICAL_ADVANCEMENT;   < Use final rate for modeled vertical RA. >

        < Store data in RAPP table, if appropriate. >

        IF (it is time for an entry in the RAPP table)
            THEN store vertical position and velocity in RAPP table entry
                    for 'negatives' level;

        TIME = TIME + TIMINT;

    ENDREPEAT;

END vertical_only_modeling_of_maneuver_period;
```

```
--------------------------------------------------------------------------------------

PROCESS vertical_only_modeling_of_maneuver_period;

    FLT (TIME, ZDF);

    CALL FINAL_VERTICAL_RATE_DETERMINATION
            IN (ZD, SVECT.VSQ, VERTRA)
            OUT (ZDF);

    TIME = TIMINT / 2;    < Use time at middle of each interval. >

    REPEAT WHILE (TIME LE TNVRAM);

        CALL VERTICAL_ADVANCEMENT
                IN (ZDF, DELINT)
                INOUT (NVGEOM.ver);

        IF (it is time for an entry in the RAPP table)
            THEN store vertical position and velocity in RAPP table entry
                    for 'negatives' level;

        TIME = TIME + TIMINT;

    ENDREPEAT;

END vertical_only_modeling_of_maneuver_period;
```

```
--------------------------------------------------------------------------

ROUTINE PSEP_MATRIX_GENERATOR
   IN (State vectors for two aircraft,
       indication of which aircraft are to be maneuvered,
       directional sense of vertical resolution advisories if both
          aircraft are to be maneuvered)
   OUT (Pointer to predicted separation matrices,
        RAPP table for each aircraft to be maneuvered);

     < This routine generates the predicted separation matrices and RAPP table
       entries for a conflict by modeling the horizontal and vertical flight paths of
       the two aircraft. >

     Access predicted separation matrices;
     Access conflict table entries (if any) for both aircraft
        via pointers in state vectors;
     Obtain previous resolution advisores (if any) for both aircraft
        from conflict table entries;

     Initialize positions and velocities to current values;
     PERFORM modeling_of_delay_period;

     PERFORM vertical_level_selection;
     PERFORM horizontal_path_selection;
     PERFORM maneuver_time_calculation;

     PERFORM maneuver_modeling;

     PERFORM vertical_convergence_checks;
     PERFORM horizontal_convergence_checks;
     PERFORM three_dimensional_convergence_checks;

END PSEP_MATRIX_GENERATOR;
```

```
-----------------------------------------------------------------------

ROUTINE PSEP_MATRIX_GENERATOR
   IN (SVECT1, SVECT2, RSPND1, RSPND2, VERTRA1, VERTRA2)
   OUT (MATPTR, RAPP1, RAPP2);

     Access PSMAT via MATPTR;

     LOOP;   < Repeat for each aircraft. >

          IF (SVECT.CTE NE $NULL for this aircraft)
               THEN PHRA for this aircraft = SVECT.CTE->CTENTRY.HMAND;
                    PVRA for this aircraft = SVECT.CTE->CTENTRY.VMAND;
               ELSE PHRA for this aircraft = $NORES;
                    PVRA for this aircraft = $NORES;

     EXITIF (both aircraft processed);
     ENDLOOP;

     Initialize DELGEOM from state vectors;
     PERFORM modeling_of_delay_period;

     PERFORM vertical_level_selection;
     PERFORM horizontal_path_selection;
     PERFORM maneuver_time_calculation;

     PERFORM maneuver_modeling;

     PERFORM vertical_convergence_checks;
     PERFORM horizontal_convergence_checks;
     PERFORM three_dimensional_convergence_checks;

 END PSEP_MATRIX_GENERATOR;
```

```
--------------------------------------------------------------------------------

PROCESS modeling_of_delay_period;

     < This process models the flight paths of two aircraft during the delay period.
       The aircraft may be modeled in either linear or nonlinear flight, depending on
       turn status and any previous resolution advisories which may be in effect. >

     IF (neither aircraft has a strongly-sensed turn AND
         neither aircraft has a previous vertical resolution advisory AND
         neither aircraft has a previous TR or TL advisory)

          THEN < both aircraft are in linear flight. >
               PERFORM linear_modeling_of_delay;

          ELSE < one aircraft may be in nonlinear flight. >
               PERFORM nonlinear_modeling_of_delay;

     Initialize each element of PSEP2 matrix to PSEP2I;
     Initialize each element of HMD2 matrix to HMD2I;
     Initialize each element of VMDA matrix to VMDAI;
     Initialize each element of VMDB matrix to VMDBI;

END modeling_of_delay_period;
```

```
--------------------------------------------------------------------------------
PROCESS modeling_of_delay_period;

    IF ((SVECT.TURN NE $STRNGLFT AND SVECT.TURN NE $STRNGRT for both AC) AND
        (PVRA EQ $NORES OR PVRA EQ $NULLRES for both AC) AND
        (PHRA NE $TR AND PHRA NE $TL for both AC))

         THEN PERFORM linear_modeling_of_delay;

         ELSE PERFORM nonlinear_modeling_of_delay;

    Initialize each element of PSEP2 to PSEP2I;
    Initialize each element of HMD2 to HMD2I;
    Initialize each element of VMDA to VMDAI;
    Initialize each element of VMDB to VMDBI;

END modeling_of_delay_period;
```

```
PROCESS vertical_level_selection;

    < This process determines the vertical levels to be modeled for two aircraft
      during the maneuver period by determining the vertical rate to be achieved by
      each aircraft for each level. >

    LOOP:   < Repeat for each aircraft. >

        < Determine final vertical rate for each type of vertical maneuver. >

        PERFORM vertical_rate_determination;

        < Select the vertical levels according to which aircraft are
          to be maneuvered. >

        Select 'maintain vertical rate' for level 1;

        IF (both aircraft are to be maneuvered)
             THEN IF (sense of vertical advisories is 'climb' for this aircraft)
                       THEN Select 'climb' for level 2;
                            Select 'don't descend' for level 3;
                       ELSE Select 'descend' for level 2;
                            Select 'don't climb' for level 3;
             ELSE < only one aircraft is to be maneuvered. >
                  IF (this aircraft is the one to be maneuvered)
                       THEN Select 'descend' for level 2;
                            Select 'climb' for level 3;
                       ELSE Select 'maintain vertical rate' for level 2;
                            Select 'maintain vertical rate' for level 3;

    EXITIF (both aircraft have been processed);
    ENDLOOP;

END vertical_level_selection;
```

```
-------------------------------------------------------------------------------

PROCESS vertical_level_selection;

    LOOP;   < Repeat for each aircraft. >

        PERFORM vertical_rate_determination;

        ZDFH($LEV1) = DELGEOH.ZD;

        IF (RSPND1 EQ $TRUE AND RSPND2 EQ $TRUE)

            THEN IF (VERTRA EQ $CL for this aircraft)
                     THEN ZDFH($LEV2) = RATE.CLH;
                          ZDFH($LEV3) = RATE.DDES;
                     ELSE ZDFH($LEV2) = RATE.DES;
                          ZDFH($LEV3) = RATE.DCL;

            ELSE IF (RSPND EQ $TRUE for this aircraft)
                     THEN ZDFH($LEV2) = RATE.DES;
                          ZDFH($LEV3) = RATE.CL;
                     ELSE ZDFH($LEV2) = DELGEOH.ZD;
                          ZDFH($LEV3) = DELGEOH.ZD;

     EXITIF (both aircraft have been processed);
     ENDLOOP;

 END vertical_level_selection;
```

```
----------------------------------------------------------------------------
PROCESS horizontal_path_selection;

     < This process determines which horizontal maneuvers will be modeled for each
       aircraft during the maneuver period. >

     LOOP;   < Repeat for each aircraft. >

          The 'continue straight' path will be modeled;

          IF (this aircraft is to be maneuvered)

               THEN IF (this aircraft has a previous TL advisory)
                         THEN only the 'turn left' path will be modeled;
                    ELSEIF (this aircraft has a previous TR advisory)
                         THEN only the 'turn right' path will be modeled;
                    OTHERWISE both the 'turn left' and 'turn right' paths
                                 will be modeled;

               ELSE ;   < no other horizontal paths will be modeled. >

     EXITIF (both aircraft have been processed);
     ENDLOOP;

END horizontal_path_selection;
```

```
------------------------------------------------------------------------------------

PROCESS horizontal_path_selection;

     LOOP:   < Repeat for each aircraft. >

          SET PATH.MODEL($CSP);

          IF (RSPND EQ $TRUE for this aircraft)

               THEN IF (PHRA EQ $TL)
                         THEN SET PATH.MODEL($TLP);
                              CLEAR PATH.MODEL($TRP);
                    ELSEIF (PHRA EQ $TR)
                         THEN SET PATH.MODEL($TRP);
                              CLEAR PATH.MODEL($TLP);
                    OTHERWISE SET PATH.MODEL($TRP);
                              SET PATH.MODEL($TLP);

               ELSE CLEAR PATH.MODEL($TRP);
                    CLEAR PATH.MODEL($TLP);

     EXITIF (both aircraft have been processed);
     ENDLOOP;

END horizontal_path_selection;
```

```
--------------------------------------------------------------------------------
PROCESS maneuver_time_calculation;

    < This process determines the length of time to model each aircraft during the
      maneuver period. >

    Compute position of aircraft 2 relative to aircraft 1 (vertical
       weighted) after delay;
    Compute velocity of aircraft 2 relative to aircraft 1 (vertical
       weighted) after delay;

    IF (magnitude of relative velocity is very small)

        THEN < use slow-closing value. >
             Maneuver time = MTSC/number of aircraft being maneuvered;

        ELSE Compute time to 3-D closest approach (vertical weighted)
                after delay, using relative position and velocity;
             Maneuver time = time to closest approach + TCADEL;
             Apply a lower limit of MTLL to maneuver time;
             Apply an upper limit of MTUL to maneuver time;

    Compute time to turn the slowest maneuvered aircraft through an angle of
       TURNA1 and apply as an upper limit to maneuver time;
    IF (both aircraft are to be maneuvered)
        THEN compute time to turn both aircraft through a combined angle of
                TURNA2 and apply as an upper limit to maneuver time;

END maneuver_time_calculation;
```

```
--------------------------------------------------------------------------------

PROCESS maneuver_time_calculation;

    FLT (TCA, V2, W);

    RX = DELGEOM.ac2.X - DELGEOM.ac1.X;
    RY = DELGEOM.ac2.Y - DELGEOM.ac1.Y;
    RZ = (DELGEOM.ac2.Z - DELGEOM.ac1.Z) * VWEIGHT;
    VRX = DELGEOM.ac2.XD - DELGEOM.ac1.XD;
    VRY = DELGEOM.ac2.YD - DELGEOM.ac1.YD;
    VRZ = (DELGEOM.ac2.ZD - DELGEOM.ac1.ZD) * VWEIGHT;
    VR2 = VRX**2 + VRY**2 + VRZ**2;

    IF (VR2 LT VRTH2)

        THEN MANTM = MTSC/number of aircraft being maneuvered;

        ELSE TCA = -(RX * VRX + RY * VRY + RZ * VRZ) / VR2;
             MANTM = TCA + TCADEL;
             MANTM = MAX(MANTM, MTLL);
             MANTM = MIN(MANTM, MTUL);

    IF (RSPND1 EQ $TRUE)
        THEN IF (RSPND2 EQ $TRUE)
                THEN V2 = MIN(SVECT1.VSQ, SVECT2.VSQ);
                ELSE V2 = SVECT1.VSQ;
        ELSE V2 = SVECT2.VSQ;
    W = G * TAN(BANKA) / SQRT(VSQ);
    MANTM = MIN(MANTM, (TURNA1 / W));

    IF (RSPND1 EQ $TRUE AND RSPND2 EQ $TRUE)
        THEN W = G * TAN(BANKA) * (1 / SQRT(SVECT1.VSQ) + 1 / SQRT(SVECT2.VSQ));
             MANTM = MIN(MANTM, (TURNA2 / W));

END maneuver_time_calculation;

------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOW-LEVEL LOGIC ------------
```

```
-----------------------------------------------------------------------------------

PROCESS maneuver_modeling;

     < This process models two aircraft during the maneuver period by performing a
       fast-time simulation. >

     FLT TIME;   < local variable >

     PERFORM geometry_initialization;   < Start with post-delay values. >
     CALL COMPUTATION_OF_TURN_CONSTANTS;   < for aircraft 1 >
     CALL COMPUTATION_OF_TURN_CONSTANTS;   < for aircraft 2 >

     TIME = TIMINT/2;   < Use time at middle of each interval. >

     REPEAT WHILE (TIME LE maneuver time):

          < Advance each aircraft by TIMINT seconds, and store data in
            RAPP table, if appropriate. >

          PERFORM incremental_advancement;

          < Determine minimum separation for each combination of flight paths. >

          PERFORM separation_calculations;
          PERFORM collection_of_minimums;

          < Save the 'quick separation' matrix at the appropriate time. >

          TIME = TIME + TIMINT;
          IF (QTIME has just been reached)
               THEN save current separation values in QSEP2 matrix;

     ENDREPEAT;

END maneuver_modeling;
```

```
--------------------------------------------------------------------------------

PROCESS maneuver_modeling;

     FLT TIME;
     BIT QFLAG;

     PERFORM geometry_initialization;
     CALL COMPUTATION_OF_TURN_CONSTANTS IN (SVECT1.VSQ, TIMINT)
                                        OUT (TURCON.ac1);
     CALL COMPUTATION_OF_TURN_CONSTANTS IN (SVECT2.VSQ, TIMINT)
                                        OUT (TURCON.ac2);

     TIME = TIMINT/2;
     CLEAR QFLAG;

     REPEAT WHILE (TIME LE MANTM);

          PERFORM incremental_advancement;

          PERFORM separation_calculations;
          PERFORM collection_of_minimums;

          TIME = TIME + TIMINT;
          IF (QFLAG EQ $FALSE AND TIME GE QTIME)
               THEN QSEP2 = CURP2;
                    SET QFLAG;

     ENDREPEAT;

END maneuver_modeling;
```

```
------------------------------------------------------------------------------------
PROCESS vertical_convergence_checks;

    < This process modifies the calculated vertical miss distance for any vertical
      level where convergence is indicated at the end of the maneuver period. >

    LOOP;   < Repeat for each vertical level. >

        Determine vertical convergence for this level;

        IF (aircraft are converging vertically)
            THEN VMDA = 0 for this level;
            ELSE :   < no change >

    EXITIF (all vertical levels examined);
    ENDLOOP;

END vertical_convergence_checks;
```

```
------------------------------------------------------------------------------------
PROCESS vertical_convergence_checks;

     INT LEVEL;

     LOOP:   < Repeat for each vertical level. >

          RZ = NANGEOM.ver2(LEVEL).Z - NANGEOM.ver1(LEVEL).Z;
          VRZ = NANGEOM.ver2(LEVEL).ZD - NANGEOM.ver1(LEVEL).ZD;
          DOT = RZ * VRZ;

          IF (DOT LT 0)
               THEN VNDA(LEVEL) = 0;
               ELSE :   < no change >

     EXITIF (all vertical levels examined);
     ENDLOOP;

END vertical_convergence_checks;
```

```
PROCESS horizontal_convergence_checks;

    < This process modifies the calculated horizontal miss distance for any
      combination of horizontal flight paths where convergence is indicated at the
      end of the maneuver period. >

    LOOP:   < Repeat for each horizontal path modeled for aircraft 1. >

        LOOP:   < Repeat for each horizontal path modeled for aircraft 2. >

            < Determine horizontal convergence for this combination of
              horizontal flight paths. >

            CALL CONVERGENCE_HORIZONTAL;
            IF (aircraft are converging at maneuver time)
                 THEN IF (horizontal combination is 'straight/straight')
                           THEN < use horizontal miss-distance formula
                                  to compute HMD2. >
                                CALL MISS_DISTANCE_HORIZONTAL;
                           ELSE HMD2 = 0 for this horizontal combination;
                 ELSE :   < no change >

        EXITIF (all horizontal paths examined for aircraft 2);
        ENDLOOP;

    EXITIF (all horizontal paths examined for aircraft 1);
    ENDLOOP;

END horizontal_convergence_checks;
```

```
------------------------------------------------------------------------------

PROCESS horizontal_convergence_checks;

     INT (HPATH1, HPATH2);

     LOOP;   < Repeat for each horizontal path modeled for aircraft 1. >

          LOOP:   < Repeat for each horizontal path modeled for aircraft 2. >

               CALL CONVERGENCE_HORIZONTAL
                       IN (MANGEOM.hor1(HPATH1), MANGEOM.hor2(HPATH2))
                       INOUT (MODVBL.relative_geometry);
               IF (DOT LT 0)
                    THEN IF (HPATH1 EQ $CSP AND HPATH2 EQ $CSP)
                              THEN CALL MISS_DISTANCE_HORIZONTAL
                                         IN (MODVBL.relative_geometry)
                                         OUT (HMD2(HPATH1, HPATH2));
                              ELSE HMD2(HPATH1, HPATH2) = 0;
                    ELSE :   < no change >

          EXITIF (all horizontal paths examined for aircraft 2);
          ENDLOOP;

     EXITIF (all horizontal paths examined for aircraft 1);
     ENDLOOP;

END horizontal_convergence_checks;
```

```
------------------------------------------------------------------------------

PROCESS three_dimensional_convergence_checks;

     < This process modifies the calculated 3-D miss distance for any combination
       of flight paths where 3-D convergence is indicated at the end of the
       maneuver period. >

     LOOP;   < Repeat for each horizontal path modeled for aircraft 1. >
          LOOP;   < Repeat for each horizontal path modeled for aircraft 2. >

               LOOP;    < Repeat for each vertical level. >

                    < Determine 3-D convergence for this combination of
                      flight paths. >

                    CALL CONVERGENCE_3D;
                    IF (aircraft are converging at maneuver time)
                         THEN IF (horizontal combination is 'straight/straight')
                                   THEN < use 3-D miss-distance formula
                                          to compute PSEP2. >
                                        CALL MISS_DISTANCE_3D;
                                        VMDB for this vertical level =
                                           vertical component of PSEP2;
                                   ELSE PSEP2 = 0;
                                        VMDB for this vertical level = 0;
                         ELSE :    < no change >

               EXITIF (all vertical levels examined);
               ENDLOOP;

          EXITIF (all horizontal paths examined for aircraft 2);
          ENDLOOP;
     EXITIF (all horizontal paths examined for aircraft 1);
     ENDLOOP;

END three_dimensional_convergence_checks;
```

```
--------------------------------------------------------------------------------
PROCESS three_dimensional_convergence_checks;

    INT (HPATH1, HPATH2, LEVEL);

    LOOP:   < Repeat for each horizontal path modeled for aircraft 1. >

        LOOP:   < Repeat for each horizontal path modeled for aircraft 2. >

            LOOP:   < Repeat for each vertical level. >

                CALL CONVERGENCE_3D
                        IN (MANGEOM.hor1(HPATH1), MANGEOM.ver1(LEVEL),
                            MANGEOM.hor2(HPATH2), MANGEOM.ver2(LEVEL))
                        OUT (MODVBL.relative_geometry);
                IF (DOT LT 0)
                    THEN IF (HPATH1 EQ $CSP AND HPATH2 EQ $CSP)
                            THEN CALL MISS_DISTANCE_3D
                                        IN (MODVBL.relative_geometry,$TRUE)
                                        OUT (PSEP2(HPATH1,HPATH2,LEVEL),
                                             VMDB(LEVEL));
                            ELSE PSEP2(HPATH1, HPATH2, LEVEL) = 0;
                                 VMDB(LEVEL) = 0;
                    ELSE :  < no change >

            EXITIF (all vertical levels examined);
            ENDLOOP;

        EXITIF (all horizontal paths examined for aircraft 2);
        ENDLOOP;

    EXITIF (all horizontal paths examined for aircraft 1);
    ENDLOOP;

END three_dimensional_convergence_checks;
```

```
--------------------------------------------------------------------------------------

PROCESS addition_to_RAPP_table:

    < This process stores projected positions and velocities in the RAPP table for
      each aircraft. >

    LOOP;   < Repeat for each vertical level. >

        Store vertical position and velocity for this level in RAPP table entry;

    EXITIF (all vertical levels selected);
    ENDLOOP;

    LOOP;   < Repeat for each horizontal path modeled for this aircraft. >

        Store horizontal position and velocity for this path in RAPP table entry;

    EXITIF (all horizontal paths selected for this aircraft);
    ENDLOOP;

END addition_to_RAPP_table;
```

```
PROCESS addition_to_RAPP_table;

     LOOP:   < Repeat for each vertical level. >

          Store vertical position and velocity for this level in RAPP table entry;

     EXITIF (all vertical levels selected);
     ENDLOOP;

     LOOP:   < Repeat for each horizontal path modeled for this aircraft. >

          Store horizontal position and velocity for this path in RAPP table entry;

     EXITIF (all horizontal paths selected for this aircraft);
     ENDLOOP;

END addition_to_RAPP_table;
```

```
PROCESS collection_of_minimums;

    < This process is performed at each time step during the maneuver period to
      save the minimum separation values. >

    LOOP:  < Repeat for each vertical level. >
         IF (current vertical separation for this level LT previous minimum)
              THEN save current separation as new minimum in VHDA;
    EXITIF (all vertical levels processed);
    ENDLOOP;

    LOOP:  < Repeat for each horizontal path modeled for aircraft 1. >
         LOOP:   < Repeat for each horizontal path modeled for aircraft 2. >

              IF (current range for this horizontal combination LT
                  previous minimum)
                   THEN save current range as new minimum in HHD2;

              LOOP:  < Repeat for each vertical level. >
                   IF (current slant range for this combination of
                       flight paths LT previous minimum)
                        THEN Save current slant range as new minimum in
                             3-D PSEP2 array;
                             IF (horizontal combination is 'straight/straight')
                                  THEN save current vertical separation
                                       in VHDB for this level;
              EXITIF (all vertical levels processed);
              ENDLOOP;

         EXITIF (all horizontal paths processed for aircraft 2);
         ENDLOOP;
    EXITIF (all horizontal paths processed for aircraft 1);
    ENDLOOP;

END collection_of_minimums;
```

```
---------------------------------------------------------------------------------
PROCESS collection_of_minimums;

     INT (HPATH1, HPATH2, LEVEL);

     LOOP:   < Repeat for each vertical level. >
          IF (CURV(LEVEL) LT VMDA(LEVEL))
               THEN VMDA(LEVEL) = CURV(LEVEL);
     EXITIF (all vertical levels processed);
     ENDLOOP;

     LOOP:   < Repeat for each horizontal path modeled for aircraft 1. >

          LOOP:   < Repeat for each horizontal path modeled for aircraft 2. >

               IF (CURH2(HPATH1, HPATH2) LT HMD2(HPATH1, HPATH2))
                    THEN HMD2(HPATH1, HPATH2) = CURH2(HPATH1, HPATH2);

               LOOP:   < Repeat for each vertical level. >
                    IF (CURP2(HPATH1, HPATH2, LEVEL) LT
                        PSEP2(HPATH1, HPATH2, LEVEL))
                         THEN PSEP2(HPATH1, HPATH2, LEVEL) =
                                   CURP2(HPATH1, HPATH2, LEVEL);
                              IF (HPATH1 EQ SCSP AND HPATH2 EQ SCSP)
                                   THEN VMDB(LEVEL) = CURV(LEVEL);
               EXITIF (all vertical levels processed);
               ENDLOOP;

          EXITIF (all horizontal paths processed for aircraft 2);
          ENDLOOP;

     EXITIF (all horizontal paths processed for aircraft 1);
     ENDLOOP;

END collection_of_minimums;

------------ RESOLUTION ADVISORIES EVALUATION ROUTINE LOW-LEVEL LOGIC ------------
```

```
--------------------------------------------------------------------------------

PROCESS geometry_initialization;

    < This process initializes the position and velocity variables for each aircraft
      prior to the modeling of the maneuver period. >

    LOOP:   < Repeat for each aircraft. >

        LOOP:   < Repeat for each horizontal path modeled for this aircraft. >
             Initial horizontal position and velocity for this path = projected
                horizontal position and velocity at end of delay period;
        EXITIF (each horizontal path has been selected);
        ENDLOOP;

        LOOP:   < Repeat for each vertical level. >
             Initial altitude and vertical rate for this level = projected
                altitude and vertical rate at end of delay period;
        EXITIF (each vertical level has been selected);
        ENDLOOP;

    EXITIF (both aircraft have been processed);
    ENDLOOP;

END geometry_initialization;
```

```
-------------------------------------------------------------------------------------
PROCESS geometry_initialization;

     INT (HPATH, LEVEL);

     LOOP;   < Repeat for each aircraft. >

          LOOP;   < Repeat for each horizontal path modeled for this aircraft. >
               NANGEOM.hor(HPATH) = DELGEOM.hor;
          EXITIF (each horizontal path has been selected);
          ENDLOOP;

          LOOP;   < Repeat for each vertical level. >
               NANGEOM.ver(LEVEL) = DELGEOM.ver;
          EXITIF (each vertical level has been selected);
          ENDLOOP;

     EXITIF (both aircraft have been processed);
     ENDLOOP;

END geometry_initialization;
```

```
--------------------------------------------------------------------------------

PROCESS incremental_advancement;

     < This process advances each aircraft incrementally at each time step during the
       modeling of the maneuver period. >

     LOOP:   < Repeat for each aircraft. >

          < Advance aircraft vertically. >

          LOOP:   < Repeat for each vertical level. >
               CALL VERTICAL_ADVANCEMENT;
          EXITIF (all vertical levels processed);
          ENDLOOP;

          < Advance aircraft horizontally. >

          CALL CONTINUE_STRAIGHT;   < 'Straight' path always modeled. >
          IF ('turn left' is being modeled for this aircraft)
               THEN CALL TURN_LEFT;
          IF ('turn right' is being modeled for this aircraft)
               THEN CALL TURN_RIGHT;

          < Add data to RAPP table at the appropriate time. >

          IF (this aircraft is being maneuvered AND
              it is time for an entry in the RAPP table)
               THEN PERFORM addition_to_RAPP_table;

     EXITIF (both aircraft processed);
     ENDLOOP;

END incremental_advancement;
```

```
------------------------------------------------------------------------------------
PROCESS incremental_advancement;

     INT (HPATH, LEVEL);

     LOOP;   < Repeat for each aircraft. >

          LOOP;   < Repeat for each vertical level. >
               CALL VERTICAL_ADVANCEMENT IN (RATE.ZDPH(LEVEL), TIMINT)
                                           INOUT (HANGEOM.ver(LEVEL));
          EXITIF (all vertical levels processed);
          ENDLOOP;

          CALL CONTINUE_STRAIGHT IN (TIMINT)
                                 INOUT (HANGEOM.hor($CSP));
          IF (PATH.MODEL($TLP) EQ $TRUE)
               THEN CALL TURN_LEFT IN (TURCON)
                                  INOUT (HANGEOM.hor($TLP));
          IF (PATH.MODEL($TRP) EQ $TRUE)
               THEN CALL TURN_RIGHT IN (TURCON)
                                  INOUT (HANGEOM.hor($TRP));

          IF (RSPND EQ $TRUE for this aircraft AND
              it is time for an entry in the RAPP table)
              THEN PERFORM addition_to_RAPP_table;

     EXITIF (both aircraft processed);
     ENDLOOP;

END incremental_advancement;
```

```
--------------------------------------------------------------------------------
PROCESS linear_modeling_of_delay;

    < This process models the delay period by projecting each aircraft straight for
      DELAY seconds. >

    LOOP;   < Repeat for each aircraft. >
         CALL VERTICAL_ADVANCMENT;   < Use current vertical rate. >
         CALL CONTINUE_STRAIGHT;
    EXITIF (both aircraft processed);
    ENDLOOP;

    CALL CONVERGENCE_3D;
    IF (aircraft are converging in 3-D after delay)
         THEN PSEP2I = 3-D separation after delay;
              VMDBI = vertical separation after delay;
         ELSE < use 3-D miss-distance formula to compute PSEP2I. >
              CALL MISS_DISTANCE_3D;
              VMDBI = vertical component of PSEP2I;

    CALL CONVERGENCE_HORIZONTAL;
    IF (aircraft are converging in range after delay)
         THEN HMD2I = range after delay;
         ELSE < use horizontal miss-distance formula to compute HMD2I. >
              CALL MISS_DISTANCE_HORIZONTAL;

    IF (aircraft are diverging vertically before delay)
         THEN VMDAI = vertical separation before delay;
    ELSEIF (aircraft are not diverging vertically after delay)
         THEN VMDAI = vertical separation after delay;
    OTHERWISE VMDAI = 0;

END linear_modeling_of_delay;
```

```
------------------------------------------------------------------------------

PROCESS linear_modeling_of_delay;

     LOOP;   < Repeat for each aircraft. >
          CALL VERTICAL_ADVANCMENT IN (DELGEOM.ZD for this AC, DELAY)
                                   INOUT (DELGEOM.ver for this AC);
          CALL CONTINUE_STRAIGHT IN (DELAY)
                                 INOUT (DELGEOM.hor for this AC);
     EXITIF (both aircraft processed);
     ENDLOOP;

     CALL CONVERGENCE_3D IN (DELGEOM.hor1, DELGEOM.ver1, DELGEOM.hor2, DELGEOM.ver2)
                         OUT (MODVBL.relative_geometry);
     IF (DOT LT 0)
          THEN PSEP2I = RX**2 + RY**2 + RZ**2;
               VMDBI = ABS(DELGEOM.ver2.Z - DELGEOM.ver1.Z);
          ELSE CALL MISS_DISTANCE_3D
                         IN (MODVBL.relative_geometry, $TRUE)
                         OUT (PSEP2I, VMDBI);
     CALL CONVERGENCE_HORIZONTAL IN (DELGEOM.hor1, DELGEOM.hor2)
                                 OUT (MODVBL.relative_geometry);
     IF (DOT LT 0)
          THEN HMD2I = RX**2 + RY**2;
          ELSE CALL MISS_DISTANCE_HORIZONTAL
                         IN (MODVBL.relative_geometry)
                         OUT (HMD2I);
     DOT = (SVECT2.Z - SVECT1.Z) * (SVECT2.ZD - SVECT1.ZD);
     IF (DOT GT 0)
          THEN VMDAI = ABS(SVECT2.Z - SVECT1.Z);
          ELSE DOT = RZ * VRZ;
               IF (DOT LE 0)
                    THEN VMDAI = ABS(RZ);
                    ELSE VMDAI = 0;

END linear_modeling_of_delay;
```

```
-------------------------------------------------------------------------------------------------
PROCESS nonlinear_advancement;

     < This process advances each aircraft incrementally at each time step during
       nonlinear modeling of the delay period. >

     LOOP:   < Repeat for each aircraft. >

          < Advance aircraft vertically. >

          IF (last half of delay period)
               THEN < respond to any previous vertical advisories. >
                    CALL VERTICAL_ADVANCEMENT;
               ELSE < advance at current vertical rate. >
                    CALL VERTICAL_ADVANCEMENT;

          < Advance aircraft horizontally. >

          IF (first half of delay)
               THEN < model any sensed turns. >
                    IF (strong left turn sensed for aircraft)
                         THEN CALL TURN_LEFT;
                    ELSEIF (strong right turn sensed for aircraft)
                         THEN CALL TURN_RIGHT;
                    OTHERWISE CALL CONTINUE_STRAIGHT;
               ELSE < model response to any previous resolution advisories. >
                    IF (previous TL advisory)
                         THEN CALL TURN_LEFT;
                    ELSEIF (previous TR advisory)
                         THEN CALL TURN_RIGHT;
                    OTHERWISE CALL CONTINUE_STRAIGHT;

     EXITIF (both aircraft advanced);
     ENDLOOP;

END nonlinear_advancement;
----------  RESOLUTION ADVISORIES EVALUATION ROUTINE HIGH-LEVEL LOGIC  ------------
```

```
------------------------------------------------------------------------------

PROCESS nonlinear_advancement;

     LOOP:   < Repeat for each aircraft. >

          IF (TIME GE DELAY/2)
               THEN CALL VERTICAL_ADVANCEMENT IN (RATE.ZDFD, DELINT)
                                                INOUT (DELGEOM.ver);
               ELSE CALL VERTICAL_ADVANCEMENT IN (DELGEOM.ZD, DELINT)
                                                INOUT (DELGEOM.ver);

          IF (TIME LT DELAY/2)

               THEN IF (SVECT.TURN EQ $STRNGLFT)
                          THEN CALL TURN_LEFT IN (TURCON)
                                               INOUT (DELGEOM.hor);
                    ELSEIF (SVECT.TURN EQ $STRNGRGT)
                          THEN CALL TURN_RIGHT IN (TURCON)
                                                INOUT (DELGEOM.hor);
                    OTHERWISE CALL CONTINUE_STRAIGHT IN (DELINT)
                                                       INOUT (DELGEOM.hor);

               ELSE IF (PHRA EQ $TL)
                          THEN CALL TURN_LEFT IN (TURCON)
                                               INOUT (DELGEOM.hor);
                    ELSEIF (PHRA EQ $TR)
                          THEN CALL TURN_RIGHT IN (TURCON)
                                                INOUT (DELGEOM.hor);
                    OTHERWISE CALL CONTINUE_STRAIGHT IN (DELINT)
                                                       INOUT (DELGEOM.hor);

     EXITIF (both aircraft advanced);
     ENDLOOP;

END nonlinear_advancement;
```

```
--------------------------------------------------------------------------------
PROCESS nonlinear_delay_preparations;

    < This process computes the turn constants and final vertical rates for each
      aircraft in preparation for nonlinear modeling of the delay period. >

    LOOP;   < Repeat for each aircraft. >

        < Compute final vertical rate for delay period. >

        CALL FINAL_VERTICAL_RATE_DETERMINATION;

        < Compute turn constants for delay period. >

        CALL COMPUTATION_OF_TURN_CONSTANTS;

    EXITIF (both aircraft processed);
    ENDLOOP;

END nonlinear_preparations;
```

```
--------------------------------------------------------------------------------
PROCESS nonlinear_delay_preparations:

     FLT (ZDMIN, ZDMAX);

     LOOP:   < Repeat for each aircraft. >

          CALL FINAL_VERTICAL_RATE_DETERMINATION
                 IN (SVECT.ZD, SVECT.VSQ, PVRA)
                 OUT (RATE.ZDFD for this aircraft);

          CALL COMPUTATION_OF_TURN_CONSTANTS
                 IN (SVECT.VSQ, DELINT)
                 OUT (TURCON for this aircraft);

     EXITIF (both aircraft processed);
     ENDLOOP;

END nonlinear_preparations;
```

```
--------------------------------------------------------------------------------
PROCESS nonlinear_modeling_of_delay;

     < This process models the delay period nonlinearly by performing a
       fast-time simulation. >

     FLT TIME;   < local variable >

     PERFORM nonlinear_delay_preparations;

     PSEP2I = 3-D separation (vertical weighted) before delay;
     HMD2I = range before delay;
     VMDAI = vertical separation before delay;
     VMDBI = vertical separation before delay;
     TIME  = 0;

     REPEAT UNTIL (TIME GE DELAY);

          < Advance each aircraft by DELINT seconds. >

          PERFORM nonlinear_advancement;

          < Save minimum separation values. >

          Compute current 3-D, horizontal, and vertical separation;
          IF (current vertical-weighted 3-D separation LT PSEP2I)
               THEN PSEP2I = current vertical-weighted 3-D separation;
                    VMDBI = current vertical separation (unweighted);
          HMD2I  = MIN(HMD2I, current range);
          VMDAI = MIN(VMDAI, current vertical separation);

          TIME  = TIME + DELINT;

     ENDREPEAT;

END nonlinear_modeling_of_delay;
```

AD-A104 148 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-2 FAA-RD-81-45-2 NL

5 OF 7
AD
A104148

```
-----------------------------------------------------------------------------
PROCESS nonlinear_modeling_of_delay;

     FLT (TIME, P2, H2, V);

     PERFORM nonlinear_delay_preparations;

     HMD2I = (SVECT2.X - SVECT1.X)**2 + (SVECT2.Y - SVECT1.Y)**2;
     VMDAI = ABS(SVECT2.Z - SVECT1.Z);
     VMDBI = VMDAI;
     PSEP2I = HMD2I + (VMDAI * VWEIGHT)**2;

     TIME  = 0;

     REPEAT UNTIL (TIME GE DELAY);

          PERFORM nonlinear_advancement;

          H2 = (DELGEOM.hor2.X - DELGEOM.hor1.X)**2 +
                    (DELGEOM.hor2.Y - DELGEOM.hor2.Y)**2;
          V = ABS(DELGEOM.ver2.Z - DELGEOM.ver1.Z);
          P2 = H2 + (V * VWEIGHT)**2;
          IF (P2 LT PSEP2I)
               THEN PSEP2I = P2;
                    VMDBI = V;
          HMD2I = MIN(HMD2I, H2);
          VMDAI = MIN(VMDAI, V);

          TIME  = TIME + DELINT;

     ENDREPEAT;

END nonlinear_modeling_of_delay;
```

```
--------------------------------------------------------------------------------------

PROCESS separation_calculations;

     < This process computes the vertical, horizontal, and 3-D separation values at
       time step during the maneuver period. >

     LOOP:   < Repeat for each vertical level. >
          Compute and save vertical separation for this level;
     EXITIF (all vertical levels processed);
     ENDLOOP;

     LOOP;   < Repeat for each horizontal path modeled for aircraft 1. >

          LOOP;   < Repeat for each horizontal path modeled for aircraft 2. >

               Compute and save horizontal separation (range) for this
                  combination of horizontal paths;

               LOOP;   < Repeat for each vertical level. >
                    Compute and save 3-D separation (slant range, vertical
                       weighted) for this combination of flight paths;
               EXITIF (all vertical levels processed);
               ENDLOOP;

          EXITIF (all horizontal paths processed for aircraft 2);
          ENDLOOP;

     EXITIF (all horizontal paths processed for aircraft 1);
     ENDLOOP:

END separation_calculations;
```

```
-------------------------------------------------------------------------------
PROCESS separation_calculations;

    INT (LEVEL, HPATH1, HPATH2);

    LOOP;   < Repeat for each vertical level. >
        CURV(LEVEL) = ABS(MANGEOM.ver2(LEVEL).Z - MANGEOM.ver1(LEVEL).Z);
    EXITIF (all vertical levels processed);
    ENDLOOP;

    LOOP;   < Repeat for each horizontal path modeled for aircraft 1. >

        LOOP;   < Repeat for each horizontal path modeled for aircraft 2. >

            CURH2(HPATH1, HPATH2) =
               (MANGEOM.hor2(HPATH2).X - MANGEOM.hor1(HPATH1).X)**2+
               (MANGEOM.hor2(HPATH2).Y - MANGEOM.hor1(HPATH1).Y)**2;

            LOOP;   < Repeat for each vertical level. >
                CURP2(HPATH1, HPATH2, LEVEL) = CURH2(HPATH1, HPATH2)
                   + (CURV(LEVEL) * VWEIGHT)**2;
            EXITIF (all vertical levels processed);
            ENDLOOP;

        EXITIF (all horizontal paths processed for aircraft 2);
        ENDLOOP;

    EXITIF (all horizontal paths processed for aircraft 1);
    ENDLOOP;

END separation_calculations;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_rate_determination;

    < This process determines the final vertical rate to be achieved by an aircraft
      in response to each vertical resolution advisory which may be modeled during
      the maneuver period. >

    < Select the required rates. >

    IF (this is a 'fast' aircraft)
        THEN 'Climb' rate = ZDUPF;
             'Descend' rate = -ZDDWNF;
        ELSE < this is a 'slow' aircraft. >
             'Climb' rate = ZDUPS;
             'Descend' rate = -ZDDWNS;

    < Use the delayed vertical rate if it already exceeds the required rate. >

    'Climb' rate = MAX('climb' rate, vertical rate after delay);
    'Don't descend' rate = MAX(0, vertical rate after delay);
    'Descend' rate = MIN('Descend' rate, vertical rate after delay);
    'Don't climb' rate = MIN(0, vertical rate after delay);

    < Change 'descend' to 'don't climb' if aircraft is too low. >

    IF (current altitude LT terrain altitude from state vector + ATERN)
        THEN 'Descend' rate = 'don't climb' rate;

END vertical_rate_determination;
```

```
-------------------------------------------------------------------------------

PROCESS vertical_rate_determination;

    IF (SVECT.VSQ GT VTHSQ)
        THEN RATE.CLM for this aircraft = ZDUPF;
             RATE.DES for this aircraft = -ZDDWNF;
        ELSE RATE.CLM for this aircraft = ZDUPS;
             RATE.DES for this aircraft = -ZDDWNS;

    RATE.CLM = MAX(RATE.CLM, DELGEOM.ZD);
    RATE.DDES = MAX(0, DELGEOM.ZD);
    RATE.DES = MIN(RATE.DES, DELGEOM.ZD);
    RATE.DCL = MIN(0, DELGEOM.ZD);

    IF (SVECT.Z LT SVECT.TERALT + ATERN)
        THEN RATE.DES = RATE.DCL;

END vertical_rate_determination;
```

```
------------------------------------------------------------------------------

ROUTINE RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION
   IN (Resolution advisories for two aircraft, aircraft state vectors)
   OUT (Closest 3-D separation for the pair);

     < This routine models one set of resolution advisories for a pair of aircraft to
       determine the 3-D miss distance which those advisories will produce. >

     Access conflict table entries for both aircraft via pointers
        in state vectors;
     Obtain previous resolutions advisories for both aircraft from
        conflict table entries;
     Initialize positions and velocities to current values;

     PERFORM one_path_modeling_of_delay_period;

     PERFORM maneuver_time_calculation;   < Same as in PSEP_MATRIX_GENERATOR >

     LOOP;   < Repeat for each aircraft. >
          CALL FINAL_VERTICAL_RATE_DETERMINATION;   < Use vertical RA to be modeled >
     EXITIF (both aircraft processed);
     ENDLOOP;

     PERFORM one_path_maneuver_modeling;

     PERFORM one_path_3D_convergence_check;

END RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION;
```

```
------------------------------------------------------------------------------------

ROUTINE RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION
   IN (HRA1, VRA1, HRA2, VRA2, SVECT1, SVECT2)
   OUT (PSEP2X);

     LOOP;   < Repeat for each aircraft. >

          IF (SVECT.CTE NE $NULL for this aircraft)
               THEN PHRA for this aircraft = SVECT.CTE->CTENTRY.HMAND;
                    PVRA for this aircraft = SVECT.CTE->CTENTRY.VMAND;
               ELSE PHRA for this aircraft = $NORES;
                    PVRA for this aircraft = $NORES;

          Initialize position and velocity components in DELGEOM to values
             in SVECT for this aircraft.

     EXITIF (both aircraft processed);
     ENDLOOP;

     PERFORM one_path_modeling_of_delay_period;

     PERFORM maneuver_time_calculation;   < Same as in PSEP_MATRIX_GENERATOR >

     LOOP;   < Repeat for each aircraft. >
          CALL FINAL_VERTICAL_RATE_DETERMINATION
                  IN (DELGEOM.ZD, SVECT.VSQ, VRA)
                  OUT (RATE.ZDFD for this aircraft);
     EXITIF (both aircraft processed);
     ENDLOOP;

     PERFORM one_path_maneuver_modeling;

     PERFORM one_path_3D_convergence_check;

END RESOLUTION_ADVISORY_MODELING_FOR_PREDICTED_SEPARATION;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_modeling_of_delay_period;

    < This process models the flight paths of two aircraft during the delay period,
      prior to their responding to a single set of resolution advisories.
      The aircraft may be modeled in linear or nonlinear flight during
      the delay period. >

    IF (neither aircraft has a strongly-sensed turn AND
        neither aircraft has a previous vertical resolution advisory AND
        neither aircraft has a previous TR or TL advisory)

       THEN < both aircraft are in linear flight. >
            PERFORM one_path_linear_modeling_of_delay;

       ELSE < one aircraft may be in nonlinear flight. >
            PERFORM one_path_nonlinear_modeling_of_delay;

END one_path_modeling_of_delay_period;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_modeling_of_delay_period;

     IF ((SVECT.TURN NE $STRNGLFT AND SVECT.TURN NE $STRNGRGT for both aircraft) AND
         PVRA EQ $NORES for both aircraft AND
         (PHRA NE $TR AND PHRA NE $TL for both aircraft))

          THEN PERFORM one_path_linear_modeling_of_delay;

          ELSE PERFORM one_path_nonlinear_modeling_of_delay;

END one_path_modeling_of_delay_period;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_maneuver_modeling;

    < This process models two aircraft as responding to a single set of resolution
      advisories during the maneuver period, by performing a fast-time simulation. >

    FLT TIME;   < local variable >

    IF (resolution advisory to be modeled for aircraft 1 includes TL or TR)
        THEN CALL COMPUTATION_OF_TURN_CONSTANTS;   < for aircraft 1 >

    IF (resolution advisory to be modeled for aircraft 2 includes TL or TR)
        THEN CALL COMPUTATION_OF_TURN_CONSTANTS;   < for aircraft 2 >

    TIME = TIMINT / 2;   < Use time at middle of each interval. >

    REPEAT WHILE (TIME LE maneuver time);

        < Advance each aircraft by TIMINT seconds. >

        PERFORM one_path_incremental_advancement;

        < Determine minimum separation. >

        Compute 3-D separation (slant range, vertical weighted);
        Save minimum 3-D separation;

        TIME = TIME + TIMINT;

    ENDREPEAT;

END one_path_maneuver_modeling;
```

```
--------------------------------------------------------------------------------

PROCESS one_path_maneuver_modeling;

    PLT (TIME, P2);

    IF (HRA1 EQ STL OR HRA1 EQ STR)
        THEN CALL COMPUTATION_OF_TURN_CONSTANTS
                        IN (SVECT1.VSQ, TIMINT)
                        OUT (TURCON.ac1);

    IF (HRA2 EQ STL OR HRA2 EQ STR)
        THEN CALL COMPUTATION_OF_TURN_CONSTANTS
                        IN (SVECT2.VSQ, TIMINT)
                        OUT (TURCON.ac2);

    TIME = TIMINT / 2;   < Use time at middle of each interval. >

    REPEAT WHILE (TIME LE MANTM);

        PERFORM one_path_incremental_advancement;

        P2 = (DELGEOM.hor2.X - DELGEOM.hor1.X)**2
              + (DELGEOM.hor2.Y - DELGEOM.hor1.Y)**2
              + ((DELGEOM.ver2.Z - DELGEOM.ver1.Z) * VWEIGHT)**2;
        PSEP2X = MIN(PSEP2X, P2);

        TIME = TIME + TIMINT;

    ENDREPEAT;

END one_path_maneuver_modeling;
```

```
PROCESS one_path_3D_convergence_check;

     < This process modifies the calculated 3-D separation of two aircraft,
       responding to a single set of resolution advisories, if 3-D convergence is
       indicated at the end of the maneuver period. >

     CALL CONVERGENCE_3D;
     IF (aircraft are converging at maneuver time)
          THEN IF (resolution advisory does not contain TR or TL for either aircraft)
                    THEN < use 3-D miss-distance formula. >
                         CALL MISS_DISTANCE_3D;
                    ELSE minimum 3-D separation = 0;
          ELSE :   < no change >

END one_path_3D_convergence_check;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_3D_convergence_check;

    FLT VCDUMMY;

    CALL CONVERGENCE_3D
            IN (DELGEOM.hor1, DELGEOM.ver1,
                DELGEOM.hor2, DELGEOM.ver2)
            OUT (MODVBL.relative_geometry);
    IF (DOT LT 0)
        THEN IF (HRA NE STR AND HRA NE STL for both aircraft)
                THEN CALL MISS_DISTANCE_3D
                           IN (MODVBL.relative_geometry,$FALSE)
                           OUT (PSEP2X, VCDUMMY);
                ELSE PSEP2X = 0;
        ELSE :   < no change >

END one_path_3D_convergence_check;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_incremental_advancement;

    < This process advances each aircraft incrementally at each time step during
      the maneuver period, where only one set of resolution advisories is
      being modeled. >

    LOOP:   < Repeat for each aircraft. >

        < Advance aircraft vertically. >

        CALL VERTICAL_ADVANCEMENT;

        < Advance aircraft horizontally. >

        IF (RA being modeled for this aircraft contains 'turn left')
             THEN CALL TURN_LEFT;
        ELSEIF (RA being modeled for this aircraft contains 'turn right')
             THEN CALL TURN_RIGHT;
        OTHERWISE CALL CONTINUE_STRAIGHT;

    EXITIF (both aircraft processed);
    ENDLOOP;

END one_path_incremental_advancement;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_incremental_advancement;

     LOOP:   < Repeat for each aircraft. >

          CALL VERTICAL_ADVANCEMENT
                  IN (RATE.ZDFD for this aircraft, TIMINT)
                  INOUT (DELGEOM.ver);

          IF (HRA EQ STL)
               THEN CALL TURN_LEFT IN (TURCON)
                                   INOUT (DELGEOM.hor);
          ELSEIF (HRA EQ STR)
               THEN CALL TURN_RIGHT IN (TURCON)
                                    INOUT (DELGEOM.hor);
          OTHERWISE CALL CONTINUE_STRAIGHT IN (TIMINT)
                                           INOUT (DELGEOM.hor);

     EXITIF (both aircraft processed);
     ENDLOOP;

END one_path_incremental_advancement;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_linear_modeling_of_delay

   < This process models the delay period, prior to the aircraft responding
     to a single set of advisories, by projecting each aircraft straight for
     DELAY seconds. >

   LOOP:   < Repeat for each aircraft. >

        CALL VERTICAL_ADVANCEMENT;   < Use current vertical rate. >
        CALL CONTINUE_STRAIGHT;

   EXITIF (both aircraft processed);
   ENDLOOP;

   < Perform convergence check. >

   CALL CONVERGENCE_3D;
   IF (aircraft are converging in 3-D after delay)
        THEN minimum 3-D separation so far = separation after delay;
        ELSE < use 3-D miss-distance formula to compute minimum separation. >
             CALL MISS_DISTANCE_3D;

END one_path_linear_modeling_of_delay;
```

```
---------------------------------------------------------------------------------

PROCESS one_path_linear_modeling_of_delay

    FLT VCDUMMY;

    LOOP:  < Repeat for each aircraft. >

        CALL VERTICAL_ADVANCEMENT
                IN (DELGEOM.ZD for this aircraft, DELAY)
                INOUT (DELGEOM.ver for this aircraft);
        CALL CONTINUE_STRAIGHT
                IN (DELAY)
                INOUT (DELGEOM.hor for this aircraft);

    EXITIF (both aircraft processed);
    ENDLOOP;

    CALL CONVERGENCE_3D
            IN (DELGEOM.hor1, DELGEOM.ver1,
                DELGEOM.hor2, DELGEOM.ver2)
            OUT (MODVBL.relative_geometry);
    IF (DOT LT 0)
        THEN PSEP2X = RX**2 + RY**2 + RZ**2;
        ELSE CALL MISS_DISTANCE_3D
                    IN (MODVBL.relative_geometry, $FALSE)
                    OUT (PSEP2X, VCDUMMY);

END one_path_linear_modeling_of_delay;
```

```
--------------------------------------------------------------------------------

PROCESS one_path_nonlinear_modeling_of_delay;

    < This process models the delay period nonlinearly, prior to the aircraft
      responding to a single set of resolution advisories, by performing a
      fast-time simulation. >

    FLT TIME;   < local variable >

    LOOP;   < Repeat for each aircraft. >

        CALL FINAL_VERTICAL_RATE_DETERMINATION;   < Use previous vertical RA. >
        CALL COMPUTATION_OF_TURN_CONSTANTS;   < Use DELINT time interval. >

    EXITIF (both aircraft processed);
    ENDLOOP;

    Minimum 3-D separation = slant range (vertical weighted) before delay;

    TIME = 0;

    REPEAT UNTIL (TIME GE DELAY);

        < Advance each aircraft by DELINT seconds. >

        PERFORM nonlinear_advancement;   < Same as in PSEP_MATRIX_GENERATOR >
        Compute current 3-D separation (slant range, vertical weighted);
        Save minimum slant range;

        TIME = TIME + DELINT;

    ENDREPEAT;

END one_path_nonlinear_modeling_of_delay;
```

```
--------------------------------------------------------------------------------
PROCESS one_path_nonlinear_modeling_of_delay;

    FLT (TIME, P2);

    LOOP;   < Repeat for each aircraft. >

        CALL FINAL_VERTICAL_RATE_DETERMINATION
                IN (SVECT.ZD, SVECT.VSQ, PVRA)
                OUT (RATE.ZDFD for this aircraft);
        CALL COMPUTATION_OF_TURN_CONSTANTS
                IN (SVECT.VSQ, DELINT)
                OUT (TURCON for this aircraft);

    EXITIF (both aircraft processed);
    ENDLOOP;

    PSEP2X = (SVECT2.X - SVECT1.X)**2 + (SVECT2.Y - SVECT1.Y)**2
             + ((SVECT2.Z - SVECT1.Z) * VWEIGHT)**2;

    TIME = 0;

    REPEAT UNTIL (TIME GE DELAY);

        PERFORM nonlinear_advancement;   < Same as in PSEP_MATRIX_GENERATOR >
        P2 = (DELGEOM.hor2.X - DELGEOM.hor1.X)**2
             + (DELGEOM.hor2.Y - DELGEOM.hor1.Y)**2
             + ((DELGEOM.ver2.Z - DELGEOM.ver1.Z) * VWEIGHT)**2;
        PSEP2X = MIN(PSEP2X, P2);

        TIME = TIME + DELINT;

    ENDREPEAT;

END one_path_nonlinear_modeling_of_delay;
```

```
--------------------------------------------------------------------------------------
ROUTINE TURN_LEFT
   IN (Turn constants)
   INOUT (X,Y components of position and velocity);

     < This routine models an aircraft incrementally through a left turn. >

     Compute new X,Y positional coordinates for incremental left turn;

     Compute new X,Y components of velocity for incremental left turn;

END TURN_LEFT;
```

```
-------------------------------------------------------------------------

ROUTINE TURN_LEFT
   IN (GROUP TURCON.ac)
   INOUT (GROUP GEOM.hor);

     ELT TEMP_XD;

     GEOM.X = GEOM.X - (GEOM.YD * A) + (GEOM.XD * B);
     GEOM.Y = GEOM.Y + (GEOM.XD * A) + (GEOM.YD * B);

     TEMP_XD = GEOM.XD;
     GEOM.XD = (GEOM.XD * CA) - (GEOM.YD * SA);
     GEOM.YD = (GEOM.YD * CA) + (TEMP_XD * SA);

END TURN_LEFT;
```

```
--------------------------------------------------------------------------------
ROUTINE TURN_RIGHT
   IN (Turn constants)
   INOUT (X,Y components of position and velocity);

     < This routine models an aircraft incrementally through a right turn. >

     Compute new X,Y positional coordinates for incremental right turn;

     Compute new X,Y components of velocity for incremental right turn;

END TURN_RIGHT;
```

```
--------------------------------------------------------------------------------
ROUTINE TURN_RIGHT
   IN (GROUP TURCON.ac)
   INOUT (GROUP GEOM.hor);

     FLT TEMP_XD;

     GEOM.X = GEOM.X + (GEOM.YD * A) + (GEOM.XD * B);
     GEOM.Y = GEOM.Y - (GEOM.XD * A) + (GEOM.YD * B);

     TEMP_XD = GEOM.XD;
     GEOM.XD = (GEOM.XD * CA) + (GEOM.YD * SA);
     GEOM.YD = (GEOM.YD * CA) - (TEMP_XD * SA);

END TURN_RIGHT;
```

```
ROUTINE VERTICAL_ADVANCEMENT
   IN (Aircraft final vertical rate, time interval)
   INOUT (Current aircraft altitude and vertical rate);

     < This routine models an aircraft ahead vertically for a specified interval
       of time. >

     IF (current vertical rate LT final vertical rate)
          THEN accelerate aircraft upwards using an acceleration rate of ACCELC;
     ELSEIF (current vertical rate GT final vertical rate)
          THEN accelerate aircraft downwards using an acceleration rate of ACCELD;
     OTHERWISE :   < final vertical rate already achieved. >

     Increase or decrease altitude according to vertical rate and time interval;

END VERTICAL_ADVANCEMENT;
```

```
--------------------------------------------------------------------------------
ROUTINE VERTICAL_ADVANCEMENT
   IN (ZDF, TINT)
   INOUT (GROUP GEOM.ver):

     FLT (ZDF, TINT);

     IF (GEOM.ZD LT ZDF)
          THEN GEOM.ZD = MIN(ZDF, (GEOM.ZD + ACCELC * TINT));
     ELSEIF (GEOM.ZD GT ZDF)
          THEN GEOM.ZD = MAX(ZDF, (GEOM.ZD - ACCELD * TINT));
     OTHERWISE ;   < final vertical rate already achieved. >

     GEOM.Z = GEOM.Z + GEOM.ZD * TINT;

END VERTICAL_ADVANCEMENT;
```

```
--------------------------------------------------------------------------------

ROUTINE VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION
   IN (pointer to a RADS, AC state vectors, pair record pointer)
   OUT (vertical speed limit advisories in RADS);

     IF (the aircraft are diverging horizontally)
          THEN:     <do nothing>
          ELSE LOOP:
                    Get next aircraft in pair;
               EXITIF (done both aircraft);
                    PERFORM converging_AC_check;
                    IF (AC are converging such that the AC may receive a VSL)
                         THEN IF (vertical resolution advisory for subject AC is not
                                   opposite in sense to the vertical velocity of
                                   subject AC)
                              THEN PERFORM vertical_speed_limit_calculation;
                              ELSE:
                         ELSE:
               ENDLOOP;

END VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION;
```

```
--------------------------------------------------------------------------------

ROUTINE VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION
   IN (RADSPTR, ACID1, ACID2, PREC)
   INOUT (RADSPTR.V1, RADSPTR.V2);

     BIT VSLCOMP          <compute VSL for AC if VSLCOMP is true>

     IF (the aircraft are diverging horizontally)
          THEN :     <do nothing>
          ELSE LOOP:
                    Get next aircraft in pair;
               EXITIF (done both aircraft);
                    PERFORM converging_AC_check;
                    IF (VSLCOMP EQ $TRUE)
                         THEN IF (vertical resolution advisory for subject AC is not
                                   opposite in sense to the vertical velocity
                                   of the subject AC)
                              THEN PERFORM vertical_speed_limit_calculation;
                              ELSE:
                         ELSE:
               ENDLOOP;

END VERTICAL_SPEED_LIMIT_ADVISORY_EVALUATION;
```

```
--------------------------------------------------------------------------------
PROCESS converging_AC_check;

     IF ((subject AC approaching other AC vertically) AND
               (subject AC's vertical rate LE minimum vertical speed limit
               advisory rate) AND (this AC is maneuvered))
          THEN SET flag indicating that AC are converging such that the current
                    subject AC is eligibleto receive a VSL;
          ELSE;

END converging_AC_check;
```

```
--------------------------------------------------------------------------------

PROCESS converging_AC_check;

    IF ((subject AC approaching other AC vertically) AND
            (subject AC's vertical rate LE minimum vertical speed limit
            advisory rate) AND
            RSPND EQ $TRUE))
        THEN VSLCOMP = $TRUE;
        ELSE VSLCOMP = $FALSE;

END converging_AC_check;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_speed_limit_calculation;

    Compute time to closest horizontal approach;
    Compute delay time until response to VSL is expected;
    IF (delay time is GT time to closest approach)
        THEN compute vertical speed limit;
            Truncate computed vertical speed limit to next lower
                display VSL limit;
        ELSE;

END vertical_speed_limit_calculation;
```

```
-----------------------------------------------------------------------------------

PROCESS vertical_speed_limit_calculation;

     Compute time to closest horizontal approach;
     Compute delay time until response to VSL is expected;
     IF (delay time is GT time to closest approach)
          THEN compute vertical speed limit;
               Truncate computed vertical speed limit to next lower limit;
          ELSE;

END vertical_speed_limit_calculation;
```

```
----------------------------------------------------------------------------------------

ROUTINE X_LIST_SIGNPOST_ENTRY_CALCULATION
   IN (X position)
   OUT (signpost entry on the X-List);

     <This process calculates the signpost entry point to the X-list.
      It may be used by a number of other processes.  It is being
      used here to find the entry point on the X-list for a search of
      the X-list from an AC on the EX-list for the Domino Coarse Screen
      Routine.>

     Calculate signpost = INTEGER(subject AC X position/signpost spacing);

END X_LIST_SIGNPOST_ENTRY_CALCULATION;
```

```
----------------------------------------------------------------------------

ROUTINE X_LIST_SIGNPOST_ENTRY_CALCULATION
   IN (X)
   OUT (XSGNPOST);

     <This process calculates the signpost entry point to the X-list.
      It may be used by a number of other processes.  It is being
      used here to find the entry point on the X-list for a search of
      the X-list from an AC on the EX-list for the Domino Coarse Screen
      Routine.>

     XSGNPOST = INTEGER(X / SXDP);

END X_LIST_SIGNPOST_ENTRY_CALCULATION;
```

14. MULTI-SITE RESOLUTION PROCESSING

This section describes intersite ATARS communication and the protocol involved. Communication among sites is required when aircraft are in conflict in regions serviced by more than one ATARS. The protocol involves the messages exchanged and housekeeping actions required to maintain an accurate data base.

When aircraft are in regions covered by adjacent sites, these sites coordinate to assure continuity and non-duplication of resolution service. Two means of coordination are provided in this design. Conflict Tables are exchanged using ground communication lines, where a network connection exists between two sites. This is described in Section 14.1. Elsewhere, the coordination is performed through the aircraft transponders using the Resolution Advisory Register (RAR) required for all ATARS-equipped aircraft. This register also enables coordination between ATARS and BCAS. This coordination is described in Section 14.2. See Reference 9, paragraph 3.3.2.3.1 for the detailed format of the information contained in the RAR.

The ATARS site responsible for a conflict is indicated by the ATSID variable in the Conflict Table Pair Record. This variable may also indicate that BCAS is responsible.

14.1 Conflict Table Exchange Using Ground Lines

The primary means of multi-site coordination uses ground lines, wherever these are installed. This method provides ATARS a complete and current copy of the neighboring site's Conflict Tables so that seam conflicts may be recognized and correctly resolved.

Whenever the Seam Pair Task (Section 10) recognizes a conflict containing an aircraft in a seam, it marks the pair Encounter List entry for delayed resolution. The Request and Process Remote Conflict Tables Task initiates a message through the DABS ground line network to all neighboring sites covering any part of the conflict. The request (see Table 5-3) identifies the pair of aircraft that own-site intends to resolve. This task then becomes dormant until a reply is received. The sector processing executive has the responsibility to terminate the task prematurely when it is time to begin the Master Resolution (Delayed) Task. The neighboring site returns a message to the requesting site containing zero, one, or two Conflict Tables (see Table 5-3). Two tables would be returned if the site had the two subject aircraft in unconnected conflicts. This routine then merges these Conflict Tables, so that requests on subsequent scans should always receive one Conflict Table in the reply.

When the requesting site receives each reply from a neighboring site, it performs conflict table reply processing. This process updates, adds or deletes Pair Records whose ATSID is or was set to the neighboring site's ID. This routine must be executed even if the reply contains no Conflict Table, as Pair Records may exist in own-site's copy of the Conflict Table. However, if no ground line connection exists, or if the reply is not received by the time the sector processing executive determines processing must continue, this routine is not executed. In this case the latest RAR processing update (Section 5.2) gives information on the neighboring sites' actions.

When a site receives a request for Conflict Tables, that site executes the Incoming Seam Pair Request Processing and Reply Task. This task generates the reply message and sets ATSID in the Pair Record to the requesting site's ID, unless the pair is already being resolved by own-site. It is essential that the reply message be sent as quickly as possible, so that the requesting site may process the reply before it chooses resolution advisories.

The ATCRBS aircraft which appears in an exchanged Conflict Table must be subjected to a correlation procedure by the receiving site when that ATCRBS aircraft first appears. It is necessary to perform this correlation procedure to prevent two adjacent sites from creating separate Conflict Table Entries for the same aircraft.

The site which sends a Conflict Table containing an ATCRBS aircraft will identify that aircraft with a unique ID. The receiving sites need perform the correlation only once. Thereafter, a cross-reference will link that ID to the local State Vector. The ID selected for this purpose must be one that cannot be duplicated by another remote site. For this reason, the ID is constructed by concatenating the local CTS slot number with the ID of the local site.

This cross-reference will contain entries for all ATCRBS aircraft within the local ATARS mask which the local ATARS function currently has in Conflict Tables that are being exchanged. It is identified as CREFX and is entirely unrelated to the CREFA cross-reference used in report processing. Each entry in CREFX consists of an ATCRBS ID (created by either a local or remote ATARS) and a pointer to the State Vector of this aircraft in the local CTS. For ATCRBS aircraft, the pointer in the State Vector designated ATCREF will be used as a return pointer to this entry in CREFX. Only those ATCRBS aircraft which are in seam Conflict Tables will have an entry in CREFX and have a non-null value for ATCREF.

The ATCRBS correlation procedure consists of a proximity test plus ATCRBS code check. An ATCRBS "report" is always transmitted with an ATCRBS aircraft ID in a Conflict Table. This report consists only of the current predicted range, azimuth, and altitude coordinates and the ATCRBS code. In the ATCRBS correlation, the remote range and azimuth are converted to local coordinates. The correlation procedure consists of using the X/EX-list in much the same way as in coarse screening. The proper location of the ATCRBS report in the X/EX-list is found. A search along the X/EX-list in both directions to x limits is made. All aircraft encountered are tested against y and z limits and against the ATCRBS code. The correlation procedure is successful if one and only one ATCRBS aircraft is found satisfying the requirements.

Correlation should be attempted every scan until a successful correlation occurs. Hence, the failure to correlate on the first appearance of a new ATCRBS aircraft is not fatal. An entry in REMA is created and used until a successful correlation occurs.

Two other new data structures, besides CREFX, are used to provide cross-referencing during the processing of exchanged Conflict Tables. These are the remote DABS (REMD) and remote ATCRBS (REMA) lists. A single entry on one of these lists applies to a single aircraft. The entry is a subset of the aircraft State Vector. An entry on these lists is accessed either directly with a pointer or through a cross-reference with an aircraft ID (either a DABS code or the same type of ATCRBS ID used with CREFX).

It may happen that a remote ATARS will pass a seam Conflict Table that includes one or more aircraft which are not within the local data base. The local ATARS must retain these aircraft in the Conflict Tables as place-keepers so that, when the local ATARS is required to perform conflict resolution on an aircraft in this Conflict Table which is in the local data base, an accurate Conflict Table exists. The local ATARS is not required to process these remote aircraft in any other way. Hence, the entries in REMD and REMA serve essentially as abbreviated State Vectors.

The REMFLG in the Conflict Table entry registers the current remote status of the aircraft to which that entry refers. If REMFLG is set, the ACID field in that Conflict Table entry points to an entry in REMD or REMA instead of to a State Vector. REMFLG is not transmitted in the Conflict Table message because each ATARS must determine for itself if a particular aircraft is remote.

The local ATARS determines the value to be used for NAC in the Conflict Table head of a received Conflict Table by counting the number of Conflict Table entries. This field is not transmitted in the Conflict Table Exchange Message.

14.2 Conflict Table Exchange Using RAR

Since all ATARS-equipped aircraft have a RAR, the information contained therein is always used to update and exchange conflict information. This data exchange is primary for purposes of coordination with BCAS, and for confirming that own ATARS resolution advisories were received (see Section 5.2 for both of these); and for determining the current multi-site seam status of the aircraft in geographical processing (see Section 6.2.2). When ground communication lines are installed and operating, the RAR exchange is secondary for multi-site ATARS. When no ground lines are available, the RAR becomes the primary method of coordination.

All resolution advisories sent to an aircraft are stored in the RAR (unless rejected for incompatibility). The RAR is read every scan by every ATARS site providing service to the aircraft. In this way, one site can learn of another site's action affecting aircraft in the seam. Although the conflict information exchanged this way (see Table 5-2) is less detailed than that exchanged over ground lines, it contains sufficient information to ensure selection of compatible advisories.

Every RAR column indicates the system responsible for its resolution advisories. Normally, when adjacent sites are connected, the ATARS site originally resolving a conflict continues the resolution to the conflict end. This is true even when the pair flies into a seam area where another site would normally have higher priority. However, when an aircraft leaves a site's service area, that site must release it for pairs involving this aircraft. This action is called a "handoff" in this document but is unrelated to ATC handoffs. The site releasing it sends a message to any connected neighboring sites indicated in the aircraft GEOG variable. This action gives the neighboring sites an opportunity to immediately assume responsibility for the pair. If the ground line is not available, a neighboring site takes responsibility using the rules listed in Section 10.

14.3 Pseudocode for Multi-site Resolution Processing

The pseudocode for Multi-site Resolution Processing contains two tasks: the Request and Process Remote Conflict Tables Task, and the Incoming Seam Pair Request Processing and Reply Task. These two tasks are concerned with the exchange and updating of information in the Conflict Table data structure. This pseudocode does not provide the full details of Pair Record creation, deletion or updating. These should conform to the treatment of Conflict Tables elsewhere in the document, except where otherwise specified.

PSEUDOCODE TABLE OF CONTENTS

```
-------------------------------------------------------------------------------------
TASK REQUEST_AND_PROCESS_REMOTE_CONFLICT_TABLES
   IN (Encounter list)
   OUT (messages to remote sites)
   INOUT (conflict tables);

      <For pairs requiring resolution, request conflict tables from
           connected sites that see either aircraft.>

     REPEAT WHILE (more pairs on Encounter List indicating Delayed Resolution):
          Select pair;
          Determine all connected sites that see either aircraft;
          Send CONFLICT TABLE REQUEST message for pair to these sites;
          REPEAT UNTIL (all such sites reply); <executive will terminate
                                                 loop when time is up>
               Wait for a reply to process;
               PERFORM conflict_table_reply_processing;
          ENDREPEAT;
     ENDREPEAT;

          <Pair can now go to Delayed Resolution>

END REQUEST_AND_PROCESS_REMOTE_CONFLICT_TABLES;
```

-------- REQUEST AND PROCESS REMOTE CONFLICT TABLES TASK HIGH-LEVEL LOGIC ---------

```
------------------------------------------------------------------------------------------

TASK REQUEST_AND_PROCESS_REMOTE_CONFLICT_TABLES;

   IN (Encounter list)
   OUT (messages to remote sites)
   INOUT (conflict tables);
     REPEAT WHILE (more pairs with ELENTRY.DELREQ set);
          Select pair;
          Determine all connected sites in ACID1->SVECT.GEOG OR ACID2->SVECT.GEOG;
          Send CONFLICT TABLE REQUEST MESSAGE for pair to these sites;
          REPEAT UNTIL (all such sites reply); <executive will terminate
                                              loop when time is up>
               Wait for a CONFLICT TABLE REPLY MESSAGE to process;
               PERFORM conflict_table_reply_processing;
          ENDREPEAT;
     ENDREPEAT;

END REQUEST_AND_PROCESS_REMOTE_CONFLICT_TABLES;
```

-------- REQUEST AND PROCESS REMOTE CONFLICT TABLES TASK LOW-LEVEL LOGIC ----------

```
------------------------------------------------------------------------------------------
PROCESS conflict_table_reply_processing;

     <Examine conflict tables in reply and update own data base.>

     REPEAT WHILE (any more pair records in reply);
          Select next pair record;
          IF (pair record shows replying site in control)
               THEN CALL AIRCRAFT_PAIR_IDENTIFICATION;
                    IF (both aircraft not remote)
                         THEN Create or update pair record in own conflict table;
                              Set ATSID= replying site;
                    Update track numbers;
          ELSEIF (pair record shows a non-connected site in control)
               THEN CALL AIRCRAFT_PAIR_IDENTIFICATION;
                    IF (both aircraft not remote)
                         THEN Create or update pair record in own conflict table;
                              Set ATSID= site shown in message;
                    Update track numbers;
          OTHERWISE:  <don't update if own site shown>
     ENDREPEAT;

     Delete any pair records for this site containing unknown AC;
     Update conflict table head data;

END conflict_table_reply_processing;
```

```
--------------------------------------------------------------------------------------------

PROCESS conflict_table_reply_processing;

     REPEAT WHILE (any more pair records in reply);
          Select next PREC;
          IF (PREC.ATSID=replying site)
               THEN CALL AIRCRAFT_PAIR_IDENTIFICATION
                  IN (PREC.AC1.PAC, PREC.AC2.PAC)
                  OUT (CREFX, REMA, REMD, CREFA, CREFD);
                    IF (both aircraft SVECT.RMFL not set)
                         THEN Create or update pair rec. in own conflict table;
                              PREC.ATSID= replying site;
                              CLEAR PREC.SEND1,2;
                    Update PREC.AC1.TRKID and PREC.AC2.TRKID;
          ELSEIF (PREC.ATSID= a non-connected site)
               THEN CALL AIRCRAFT_PAIR_IDENTIFICATION
                  IN (PREC.AC1.PAC, PREC.AC2.PAC)
                  OUT (CREFX, REMA, REMD, CREFA, CREFD);
                    IF (both aircraft SVECT.RMFL not set)
                         THEN Create or update pair rec. in own conflict table;
                              PREC.ATSID= site shown in msg;
                              CLEAR PREC.SEND1,2;
                    Update PREC.AC1.TRKID and PREC.AC2.TRKID;
          OTHERWISE: <don't update if own site shown>
     ENDREPEAT;

     REPEAT WHILE (any pair records with PREC.ATSID=replying site and containing
               a PREC.PAC=$UNK);
          Select next such pair record;
          CALL PAIR_RECORD_DELETION
             IN (PREC.PAC1, PREC.PAC2, pointer to PREC)
             INOUT (conflict tables, CTS);
     ENDREPEAT;
     Update CTHEAD data;

END conflict_table_reply_processing;
--------  REQUEST AND PROCESS REMOTE CONFLICT TABLES TASK LOW-LEVEL LOGIC  ----------
```

```
ROUTINE AIRCRAFT_PAIR_IDENTIFICATION
   IN (AC identifiers)
   OUT (CREFX cross-reference table);

            <Relate other site's identification to own data base.>

     REPEAT UNTIL (both aircraft processed);
          IF (aircraft described as 'unknown')
               THEN;
          ELSEIF (aircraft is DABS)
               THEN IF (DABS ID found in CREFD)
                         THEN; <CREFD points to state vector or REND entry>
                         ELSE Create REND entry and enter in CREFD;
          OTHERWISE IF (ATCRBS reference no. is in CREFX file)
                         THEN; <CREFX points to correct state vector>
                         ELSE Convert position data to local coordinates;
                              Use ATCRBS position and code data to search X/EX list
                               for sufficiently close match;
                              IF (Matching ATCRBS found on X/EX list)
                                   THEN Create CREFX entry linking other site's
                                          ATCRBS reference number to own state vector;
                                   ELSE Create or update RENA entry for aircraft;

          Select next aircraft;
     ENDREPEAT;

 END AIRCRAFT_PAIR_IDENTIFICATION;
```

```
--------------------------------------------------------------------------------

ROUTINE AIRCRAFT_PAIR_IDENTIFICATION
   IN (PREC.AC1.PAC, PREC.AC2.PAC)
   OUT (CREFX, REMA, REMD, CREFA, CREFD);

          <relate other site's identification to own data base>

     REPEAT UNTIL (both aircraft processed);
          IF (PREC.PAC EQ $UNK)
               THEN;
          ELSEIF (PREC.PAC is DABS type)
               THEN IF (DABS ID found in CREFD)
                         THEN;  <CREFD points to state vector or REMD entry>
                         ELSE Create REMD entry and enter in CREFD;
          OTHERWISE IF (ATCRBS reference no. is in CREFX file)
                         THEN;  <CREFX points to correct state vector>
                         ELSE Convert position data to local coordinates;
                              Use ATCRBS position and code data to search X/EX list
                               for sufficiently close match;
                              IF (Matching ATCRBS found on X/EX list)
                                   THEN Create CREFX entry linking other site's
                                          ATCRBS reference number to own state vector;
                                   ELSE Create or update REMA entry for aircraft;

          Select next aircraft;
      ENDREPEAT;

  END AIRCRAFT_PAIR_IDENTIFICATION;
```

```
------------------------------------------------------------------------------------------
TASK INCOMING_SEAM_PAIR_REQUEST_PROCESSING_AND_REPLY
   IN (message naming pair, requesting site ID)
    <Msg can be conflict table request, claim msg, deletion msg, handoff msg.>
   OUT (messages to remote site)
   INOUT (conflict tables);

     CALL AIRCRAFT_PAIR_IDENTIFICATION;
     IF (DELETION message)
          THEN CALL PAIR_RECORD_DELETION;
     ELSEIF (CLAIM message)
          THEN IF (pair record exists with own site in control AND own_ID GT
                   requesting site_ID)
                    THEN; <ignore claim>
                    ELSE PERFORM table_find_merge;
                         Update pair record to show requesting site in charge;
     ELSEIF (HANDOFF message)
          THEN IF (pair record exists showing sending site in charge)
                    THEN SET handoff bit in pair record;
     OTHERWISE Reply with conflict table requested;

END INCOMING_SEAM_PAIR_REQUEST_PROCESSING_AND_REPLY;
```

----- INCOMING SEAM PAIR REQUEST PROCESSING AND REPLY TASK HIGH-LEVEL LOGIC -------

```
-------------------------------------------------------------------------------------

TASK INCOMING_SEAM_PAIR_REQUEST_PROCESSING_AND_REPLY
   IN (message naming pair, requesting site ID)
  <msg can be conf. table request, claim msg, deletion msg, handoff msg>
   OUT (messages to remote site)
   INOUT (conflict tables);

     CALL AIRCRAFT_PAIR_IDENTIFICATION
        IN (PREC)
        OUT (CREFK, REMA, REMD, CREFA, CREFD);
     IF (DELETION message)
          THEN CALL PAIR_RECORD_DELETION
                  IN (ACID1, ACID2, $NULL pointer)
                  INOUT (conflict tables, CTS);
     ELSEIF (CLAIM message)
          THEN IF (pair record exists with PREC.ATSID=SYSTEM.OWNID AND
                   SYSTEM.OWNID GT requesting site_ID)
                   THEN; <ignore claim>
                   ELSE PERFORM table_find_merge;
                        PREC.ATSID=requesting site;
                        CLEAR PREC.HDOFF;
                        CLEAR PREC.SEND1,2;
     ELSEIF (HANDOFF message)
          THEN IF (pair record exists and PREC.ATSID=sending site)
                   THEN SET PREC.HDOFF;
     OTHERWISE Send CONFLICT TABLE REPLY message containing conflict table requested;

END INCOMING_SEAM_PAIR_REQUEST_PROCESSING_AND_REPLY;
```

------ INCOMING SEAM PAIR REQUEST PROCESSING AND REPLY TASK LOW-LEVEL LOGIC -------

```
--------------------------------------------------------------------------------
PROCESS table_find_merge;

    IF (aircraft in separate conflict tables)  <unknown AC always considered distinct>
        THEN Merge tables;
            Create new pair record for this pair;
    ELSEIF (aircraft in same conflict table but no pair record OR only
            one aircraft in a conflict table)
        THEN Create new pair record for pair;
    ELSEIF (neither aircraft is in a conflict table)
        THEN Create new conflict table;
            Create pair record for pair;
    OTHERWISE;  <pair record already exists>

END table_find_merge;
```

```
--------------------------------------------------------------------------------
PROCESS table_find_merge;

    IF (AC1 SVECT.CTPTR NE AC2 SVECT.CTPTR)
        THEN Merge tables;
             Create new pair record for this pair;
    ELSEIF (AC1 SVECT.CTPTR EQ AC2 SVECT.CTPTR but no pair record OR only
            one aircraft SVECT.CTPTR non-null)
        THEN Create new pair record for pair;
    ELSEIF (both AC SVECT.CTPTR's null)
        THEN Create new conflict table;
             Create pair record for pair;
    OTHERWISE: <pair rec. already exists>
    Update CTHEAD data;

END table_find_merge;
```

15. PAIR AND TRACK REMOVAL PROCESSING

The Resolution Deletion Task and the Conflict Pair Cleanup Task ensure that each conflict resolved by the local site is closed out in the proper manner when the conflict is over and that conflict data stored in the Pair Records is deleted when it is no longer needed. The State Vector Deletion Task removes aircraft from the Central Track Store when they leave the ATARS/Domino Surveillance Area or when they are no longer being adequately tracked.

15.1 Resolution Deletion Task

The Resolution Deletion Task examines the Encounter List for conflict pairs no longer requiring resolution advisories. For each such pair, this task calls the Conflict Closeout Routine to determine whether the Pair Record can be deleted. If so, the Pair Record Deletion Routine is called. If not, the PWISF flag is set in the Pair Record to indicate that the pair has been processed on the current scan.

The Conflict Closeout Routine checks for two basic conditions: (1) an aircraft has just flown out of the coverage area of the local site, and or (2) the local site has assumed responsibility for the pair, but is no longer calling for resolution advisories. When an aircraft is discovered to have flown out of coverage of the local site, all resolution advisory information pertaining to that aircraft can be cleared out of the Conflict Table; this may mean that a Pair Record involving that aircraft can be deleted immediately. For a local-responsibility conflict where resolution advisories are no longer needed, the Conflict Closeout Routine will normally place null resolution advisories in the Conflict Table. In some exceptional cases (e.g., positive advisories have not been up long enough), however, the Pair Record will be left unchanged as long as necessary. Once the aircraft have received null advisories for a conflict, the Conflict Closeout Routine will permit the Pair Record to be deleted. In instances where initial resolution advisories have not yet been selected for a conflict but the detection logic is not calling for resolution on the current scan, the Conflict Closeout Routine implements part of the two-out-of-three logic by updating POSCMD and deleting the Pair Record if appropriate.

The job of deleting a Pair Record is always performed by the Pair Record Deletion Routine. This routine is called by the Resolution Deletion and Conflict Pair Cleanup Tasks, as well as by the RAR Processing Task, the Request and Process Remote Conflict Tables Task, and the Backup Mode Initiation Process.

When a Pair Record is deleted, the Conflict Table entry for each aircraft may be simplified or deleted. It is possible at this point for a multi-aircraft Conflict Table to split into two separate Conflict Tables. It then becomes necessary to determine which aircraft belong in each of the resultant tables. This is accomplished by examining the remaining Pair Records and Conflict Table entries. This process is described conceptually in the pseudocode as the creation of three new lists (A, B, and C - lists). However, the same effect can be achieved by the manipulation of pointers, rather than by the actual creation of separate lists of aircraft and conflict pairs.

15.2 Conflict Pair Cleanup Task

The Conflict Pair Cleanup Task performs a function equivalent to the Resolution Deletion Task. The Conflict Closeout Routine and the Pair Record Deletion Routine are the primary routines called by this task. The Conflict Pair Cleanup Task serves primarily as a backup to the Resolution Deletion Task to ensure that no conflict pairs are overlooked and fail to be deleted when the data is no longer needed. This task might be needed, for instance, if a pair for which own site is responsible unexpectedly failed to pass through the coarse screen filter. Unlike the Resolution Deletion Task, the Conflict Pair Cleanup Task searches through the Pair Records for the current ATARS sector, looking for pairs which have not been processed on the current scan.

15.3 State Vector Deletion Task

This task processes aircraft on the Deletion List (not to be confused with "resolution deletion" entries on the Encounter List) and removes each aircraft State Vector from the Central Track Store if appropriate. An aircraft may be put on the Deletion List in three ways:

1. By the Track Update Process if DABS has lost surveillance contact with the aircraft.

2. By the Track Update Process if missed reports have caused the ATARS track firmness to drop below the level needed to qualify for ATARS service.

3. By the Report Processing Task if the track is seen to have left the ATARS/Domino Surveillance Area.

If the aircraft is still contained in a Conflict Table, a REMA or REMD entry is created at the time the State Vector is deleted. If ATARS has some unfinished business with the aircraft, such as a null advisory to be sent, State Vector deletion is delayed.

15.4 Pseudocode for Pair and Track Removal Processing

The pseudocode for pair and track removal is presented in this section. It should be noted that the Resolution Deletion Task and the Conflict Pair Cleanup Task call the same set of routines, which are presented once, following the Resolution Deletion Task. Also note that in the Conflict Closeout Routine, pointer (PTR) variables are used to indicate variable GROUP names; this convention does not adhere strictly to the established rules for pseudocode.

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------
<*** PARAMETERS USED IN UPDATE_SECTOR_ID ROUTINE ***>

STRUCTURE USIPARM

  GROUP values
     INT NEAR1          < Limit of "nearness" for two sectors >
     INT NEAR2          < Alternate limit of "nearness" >
     INT HALFSEC        < Half of total sectors >

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------
TASK RESOLUTION_DELETION
   IN (Encounter list, central track store)
   INOUT (Linked list of conflict tables);

     < This task examines the encounter list for conflict pairs no longer requiring
       resolution advisories from the local site.  It then ensures that each such
       conflict is closed out in the proper manner. >

     REPEAT WHILE (there are more entries in the encounter list to be examined);

          Get next entry from encounter list;
          IF (entry type EQ 'resolution deletion')
               THEN Find conflict table and pair record for this conflict;
                    CALL CONFLICT_CLOSEOUT;
                    IF (okay to delete pair record)
                         THEN CALL PAIR_RECORD_DELETION;
                         ELSE Indicate in pair record that this pair
                                 has been processed on current scan;
                              Set pointer to list of potential domino conflict
                                 aircraft to null for both aircraft;
               ELSE ;   < do not process this entry. >

     ENDREPEAT;

END RESOLUTION_DELETION;
```

------------------- RESOLUTION DELETION TASK HIGH-LEVEL LOGIC --------------------

```
--------------------------------------------------------------------------------------
TASK RESOLUTION_DELETION
   IN (Encounter list, central track store)
   INOUT (Linked list of conflict tables);

     BIT PRDELOK;

     REPEAT WHILE (there are more entries in the encounter list to be examined);

          Get next entry from encounter list;
          IF (ELENTRY.RDREQ EQ $TRUE)
               THEN Find conflict table and pair record for this conflict;
                       < as performed in PROCESS search_for_pair_record
                         in ROUTINE PAIR_RECORD_DELETION >
                    CALL CONFLICT_CLOSEOUT
                            IN (ACID1, ACID2, conflict table, pair record)
                            OUT (PRDELOK);
                    IF (PRDELOK EQ $TRUE)
                         THEN CALL PAIR_RECORD_DELETION
                                     IN (ACID1, ACID2, pointer to pair record)
                                     INOUT (Conflict tables, central track store);
                         ELSE SET PREC.PWISF;
                              PREC.ac1.INTR = $NULL;
                              PREC.ac2.INTR = $NULL;
               ELSE ;   < do not process this entry. >

     ENDREPEAT;

END RESOLUTION_DELETION;
```

-------------------- RESOLUTION DELETION TASK LOW-LEVEL LOGIC --------------------

```
-----------------------------------------------------------------------------
ROUTINE CONFLICT_CLOSEOUT
   IN (State vectors, conflict table, and pair record for a conflict)
   OUT (Indication that pair record can be deleted):

     < This routine closes out conflicts when either (1) the local site is
       responsible and is no longer calling for resolution advisories or (2) an
       aircraft in conflict has flown out of coverage. >

     IF (both aircraft are visible to own site)
          THEN PERFORM both_aircraft_visible;
     ELSEIF (one aircraft is visible to own site)
          THEN PERFORM one_aircraft_visible;
     OTHERWISE indicate that pair record can be deleted;

END CONFLICT_CLOSEOUT;
```

```
--------------------------------------------------------------------------------
ROUTINE CONFLICT_CLOSEOUT
   IN (ACID1, ACID2, conflict table, pair record)
   OUT (PRDELOK);

     BIT (VISIBLE1, VISIBLE2);
     PTR (SVECTV, acv, acnv);

     IF (SVECT1.REMFLG EQ $FALSE AND
         (SVECT1.ATSS EQ $TRUE OR SVECT1.DRATS EQ $TRUE))
          THEN SET VISIBLE1;
               SVECTV = SVECT1;
               acv = ac1;
          ELSE CLEAR VISIBLE1;
               acnv = ac1;

     IF (SVECT2.REMFLG EQ $FALSE AND
         (SVECT2.ATSS EQ $TRUE OR SVECT2.DRATS EQ $TRUE))
          THEN SET VISIBLE2;
               SVECTV = SVECT2;
               acv = ac2;
          ELSE CLEAR VISIBLE2;
               acnv = ac2;

     IF (VISIBLE1 EQ $TRUE AND VISIBLE2 EQ $TRUE)
          THEN PERFORM both_aircraft_visible;
     ELSEIF (VISIBLE1 EQ $TRUE OR VISIBLE2 EQ $TRUE)
          THEN PERFORM one_aircraft_visible;
     OTHERWISE SET PRDELOK;

END CONFLICT_CLOSEOUT;
```

```
PROCESS both_aircraft_visible;

    < This process handles the closeout of a conflict when both aircraft are
      still visible. >

    IF (the uplink of resolution advisory messages is being attempted
        to either aircraft for this conflict)
         THEN CALL UPDATE_SECTOR_ID;
              IF (handoff to another ATARS site is not being attempted AND
                  resolution advisory not null for at least one aircraft AND
                     (incompatible advisories were detected on last uplink OR
                      positive resolution advisories were not selected OR
                      positive advisories have been up long enough))
                   THEN clear resolution advisory in conflict table
                            for both aircraft;

    ELSEIF (own site is responsible for resolving this conflict AND
              initial resolution advisories have not yet been selected)
         THEN Update counter to indicate a 'miss' on this scan;
              IF (too many scans without any 'hits')
                   THEN indicate that pair record can be deleted;

    OTHERWISE :   < take no further action. >

END both_aircraft_visible;
```

```
--------------------------------------------------------------------------------

PROCESS both_aircraft_visible;

    IF (PREC.ac1.SEND EQ $TRUE OR PREC.ac2.SEND EQ $TRUE)
        THEN CALL UPDATE_SECTOR_ID
                    INOUT (ACID1, ACID2, conflict table, pair record);
            IF (PREC.HDOFF EQ $FALSE AND
                PREC.VMAN or PREC.HMAN not null for at least one aircraft AND
                  (PREC.POSCMD EQ any of $RCHSNG, $RCHDBL, $NEG,
                   $NEG, or $DOUBLE OR
                   (PREC.POSCMD EQ $POS AND
                    SYSVAR.CTIME GT PREC.TSTART + TSCMD)))
                 THEN clear resolution advisory in conflict table
                         for both aircraft;

    ELSEIF (PREC.ATSID EQ OWNID)
        THEN IF (PREC.POSCMD EQ $ONEHIT)
                 THEN PREC.POSCMD = $ONEMIS;
             ELSEIF (PREC.POSCMD EQ $ONEMIS)
                 THEN SET PRDELOK;
             OTHERWISE :

    OTHERWISE :   < take no further action. >

END both_aircraft_visible;
```

```
------------------------------------------------------------------------------

PROCESS one_aircraft_visible;

     < This process handles the closeout of a conflict when only one of the
       aircraft is still visible. >

     Clear resolution advisory in conflict table for non-visible aircraft;
     Indicate in pair record that uplink of resolution advisories to non-visible
        aircraft will no longer be attempted;

     IF (uplink of resolution advisories to visible aircraft is still being
         attempted for this conflict)

          THEN Sector ID in pair record = sector of visible aircraft;
               IF (resolution advisory NE null for visible aircraft AND
                   handoff to another ATARS site is not being attempted)
                    THEN IF (incompatible advisories were detected on last uplink)
                              THEN clear resolution advisory in conflict table
                                       for visible aircraft;
                         ELSEIF (visible aircraft is BCAS-equipped)
                              THEN indicate in pair record that uplink of resolution
                                       advisories to visible aircraft will no
                                       longer be attempted for this conflict;
                         ELSEIF (positive advisories were not selected OR
                                  positive advisories have been up long enough)
                              THEN clear resolution advisory in conflict table
                                       for visible aircraft;
                         OTHERWISE ;   < take no further action. >

          ELSE IF (resolution advisory is null for visible aircraft)
                    THEN indicate that pair record can be deleted;

END one_aircraft_visible;
```

```
------------------------------------------------------------------------------
PROCESS one_aircraft_visible;

    Clear resolution advisory in conflict table for non-visible aircraft;
    CLEAR PREC.acnv.SEND;

    IF (PREC.acv.SEND EQ $TRUE)

        THEN PREC.SECTID = SVECTV.SVSID;
            IF (PREC.VHAN or PREC.HHAN not null for visible aircraft AND
                PREC.HDOFF EQ $FALSE)
                THEN IF (PREC.POSCHD EQ $RCHSNG OR PREC.POSCHD EQ $RCHDBL)
                        THEN clear resolution advisory in conflict table
                                for visible aircraft;
                    ELSEIF (SVECTV.ATSEQ EQ $ABEQ)
                        THEN CLEAR PREC.acv.SEND;
                    ELSEIF (PREC.POSCHD NE $POS OR
                            SYSVAR.CTIME GT PREC.TSTART + TSCHD)
                        THEN clear resolution advisory in conflict table
                                for visible aircraft;
                    OTHERWISE ;   < take no further action. >

        ELSE IF (resolution advisory is null for visible aircraft)
                THEN SET PRDELOK;

END one_aircraft_visible;
```

```
--------------------------------------------------------------------------------

ROUTINE PAIR_RECORD_DELETION
   IN (IDs of two aircraft, pointer to pair record)
   INOUT (Conflict tables, central track store);

     IF (pointer to pair record EQ null)
          THEN PERFORM search_for_pair_record;

     IF (pair record exists)

          THEN PERFORM deletion_notification;
               Save aircraft IDs and pointer to pair record;
               Unlink pair record;

               LOOP;   < Repeat for each 'known' aircraft in pair. >

                    Reduce the number of conflicts for this aircraft by 1;
                    IF (there are no more conflicts involving this aircraft)
                         THEN PERFORM CTE_deletion;
                         ELSE PERFORM CTE_simplification;

               EXITIF (all 'known' aircraft in pair have been processed);
               ENDLOOP;

               Delete pair record;
               IF (no more aircraft remain in conflict table)
                    THEN unlink and delete entire conflict table;
               ELSEIF (number of aircraft remaining in conflict table LT 4)
                    THEN < conflict table cannot have split. >
                         CALL SEAM_FLAG_UPDATE;
               OTHERWISE PERFORM test_for_conflict_table_split;

          ELSE :   < take no action. >

END PAIR_RECORD_DELETION;

------------------ RESOLUTION DELETION TASK HIGH-LEVEL LOGIC --------------------
```

```
--------------------------------------------------------------------------------------

ROUTINE PAIR_RECORD_DELETION
   IN (ACID1, ACID2, PRPTR)
   INOUT (Conflict tables, central track store);

     IF (PRPTR EQ $NULL)
          THEN PERFORM search_for_pair_record;

     IF (PRPTR NE $NULL)

          THEN PERFORM deletion_notification;
               Save PREC.ac1.Pac, PREC.ac2.PAC, and PRPTR;
               Unlink pair record;

               LOOP;   < Repeat for each 'known' aircraft in pair. >
                    CTENTRY.NCON = CTENTRY.NCON - 1;
                    IF (CTENTRY.NCON EQ 0)
                         THEN PERFORM CTE_deletion;
                         ELSE PERFORM CTE_simplification;
               EXITIF (all 'known' aircraft in pair have been processed);
               ENDLOOP;

               Delete pair record;
               IF (CTHEAD.NAC EQ 0)
                    THEN unlink and delete entire conflict table;
               ELSEIF (CTHEAD.NAC LT 4)
                    THEN CALL SEAM_FLAG_UPDATE IN (Central track store)
                                                  INOUT (Conflict table);
               OTHERWISE PERFORM test_for_conflict_table_split;

          ELSE :   < take no action. >

END PAIR_RECORD_DELETION;
```

```
--------------------------------------------------------------------------------
PROCESS search_for_pair_record;

     < This process searches for a pair record, given the pointers to the state
       vectors of two aircraft (at least one of which must be available). >

     IF (aircraft 1 is unknown)
          THEN IF (aircraft 2 is not involved in any conflicts)
                    THEN no common conflict table exists;
                    ELSE common conflict table = conflict table of aircraft 2;
     ELSEIF (aircraft 2 is unknown)
          THEN IF (aircraft 1 is not involved in any conflicts)
                    THEN no common conflict table exists;
                    ELSE common conflict table = conflict table of aircraft 1;
     ELSEIF (either aircraft is not involved in any conflicts)
          THEN no common conflict table exists;
     ELSEIF (aircraft 1 and aircraft 2 are in different conflict tables)
          THEN no common conflict table exists;
     OTHERWISE < both aircraft are in the same conflict table. >
               Common conflict table = conflict table of either aircraft;

     IF (no common conflict table exists)
          THEN common pair record does not exist;
          ELSE < search conflict table for common pair record. >
               LOOP;   < Repeat for each pair record in common conflict table. >
                    IF (both aircraft are in this pair record)
                         THEN common pair record = this pair record;
                         ELSE :   < common pair record not yet found. >
               EXITIF (common pair record found OR all pair records examined);
               ENDLOOP;
               IF (no common pair record found)
                    THEN common pair record does not exist;

END search_for_pair_record;
```

```
---------------------------------------------------------------------------
PROCESS search_for_pair_record;

     PTR COMMON_CT;

     PRPTR = $NULL;

     IF (ACID1 EQ $UNK)
          THEN IF (SVECT2.CTPTR EQ $NULL)
                    THEN COMMON_CT = $NULL;
                    ELSE COMMON_CT = SVECT2.CTPTR;
     ELSEIF (ACID2 EQ $UNK)
          THEN IF (SVECT1.CTPTR EQ $NULL)
                    THEN COMMON_CT = $NULL;
                    ELSE COMMON_CT = SVECT1.CTPTR;
     ELSEIF (SVECT1.CTPTR EQ $NULL OR SVECT2.CTPTR EQ $NULL)
          THEN COMMON_CT = $NULL;
     ELSEIF (SVECT1.CTPTR NE SVECT2.CTPTR)
          THEN COMMON_CT = $NULL;
     OTHERWISE COMMON_CT = SVECT1.CTPTR;

     IF (COMMON_CT EQ $NULL)
          THEN :
          ELSE LOOP:   < Repeat for each pair record in common conflict table. >
                    IF (PREC.ac1.PAC EQ SVECT1.CTE AND PREC.ac2.PAC EQ SVECT2.CTE)
                         THEN set PRPTR to point to this pair record;
                         ELSE :   < common pair record not yet found. >
               EXITIF (PRPTR NE $NULL OR all pair records examined);
               ENDLOOP;

END search_for_pair_record;
```

```
--------------------------------------------------------------------------------
PROCESS deletion_notification;

    < This process notifies appropriate remote sites of the deletion of a pair
      record for which the local site was responsible. >

    IF (own site is responsible for this pair AND
        either aircraft is in a seam with a connected site)
         THEN Create a conflict table request message with REPLY field = 0
                and DEL field = 1;
              Send message to other connected sites indicated by
                GEOG fields in state vectors of both aircraft;

END deletion_notification;
```

```
--------------------------------------------------------------------------------

PROCESS deletion_notification;

     IF (PREC.ATSID EQ OWNID AND (SVECT1.GEOG NE OWNID OR SVECT2.GEOG NE OWNID))
          THEN Create a conflict table request message with REPLY field = 0
                  and DEL field = 1;
               Send message to other connected sites indicated by
                  SVECT1.GEOG and SVECT2.GEOG;

END deletion_notification;
```

```
--------------------------------------------------------------------------------
PROCESS CTE_deletion;

     < This process deletes an aircraft's conflict table entry. >

     Set CTPTR and CTE = null in this aircraft's state vector;
     Reduce count of aircraft in conflict table by 1;

     IF (this aircraft is remote)
          THEN IF (CREFX entry exists for this aircraft )
                    THEN delete CREFX entry;
               Unlink and delete remote list entry;

     Unlink and delete conflict table entry for this aircraft;

END CTE_deletion;
```

```
------------------------------------------------------------------------------------

PROCESS CTE_deletion:

     SVECT.CTPTR = $NULL;
     SVECT.CTE = $NULL;
     CTHEAD.NAC = CTHEAD.NAC - 1;

     IF (CTENTRY.REMFLG EQ $TRUE)
          THEN IF (CREFX entry exists for this aircraft)
                    THEN delete CREFX entry;
               Unlink and delete REMA or REMD entry;

     Unlink and delete conflict table entry for this aircraft;

END CTE_deletion;
```

```
--------------------------------------------------------------------------------
PROCESS CTE_simplification;

     < This process updates an aircraft's conflict table entry after a pair record
       involving the aircraft is deleted. >

     IF (pair record to be deleted contains horizontal RA for this aircraft)
          THEN Decrement count of horizontal resolution advisories;
               IF (count EQ 0)
                    THEN set ACIDH = null;
                    ELSE Search through remaining pair records for those
                            containing horizontal RAs for this aircraft;
                         Set ACIDH to point to one such pair record;
                         Save composite horizontal resolution advisory in HMAN;

     IF (pair record to be deleted contains vertical RA for this aircraft)
          THEN Decrement count of vertical resolution advisories;
               IF (count EQ 0)
                    THEN set ACIDV = null;
                    ELSE Search through remaining pair records for those
                            containing vertical RAs for this aircraft;
                         Set ACIDV to point to one such pair record;
                         Save composite vertical resolution advisory in VMAN;

END CTE_simplification;
```

```
-----------------------------------------------------------------------------------

PROCESS CTE_simplification;

     IF (PREC.PHMAN not null for this aircraft in pair record to be deleted)
          THEN CTENTRY.MULTH = CTENTRY.MULTH - 1;
               IF (CTENTRY.MULTH EQ 0)
                    THEN CTENTRY.ACIDH = $NULL;
                    ELSE Search through remaining pair records for those
                              containing horizontal RAs for this aircraft;
                         Set ACIDH to point to one such pair record;
                         Save composite horizontal RA in CTENTRY.HMAN;

     IF (PREC.PVMAN not null for this aircraft in pair record to be deleted)
          THEN CTENTRY.MULTV = CTENTRY.MULTV - 1;
               IF (CTENTRY.MULTV EQ 0)
                    THEN CTENTRY.ACIDV = $NULL;
                    ELSE Search through remaining pair records for those
                              containing vertical RAs for this aircraft;
                         Set ACIDV to point to one such pair record;
                         Save composite vertical RA in CTENTRY.VMAN;

END CTE_simplification;
```

```
--------------------------------------------------------------------------------
PROCESS test_for_conflict_table_split;

    < This process tests for a split in a conflict table after a pair record is
      deleted.  If the table has split, it then determines which aircraft pairs
      belong in each new table. >

    Create a linked list (list A) of all pair records remaining in conflict table;
    Select aircraft ID from first conflict table entry and place in list B;

    REPEAT UNTIL (list A is empty OR list B is empty);

        Select next aircraft ID from list B;   < subject aircraft >
        LOOP;   < Repeat for each pair in list A. >
            IF (subject aircraft is in this pair)
                THEN Remove this pair from list A;
                     Add this pair to list C;
                     Add ID of other aircraft in this pair to bottom of list B;
        EXITIF (all pairs in list A examined);
        ENDLOOP;

    ENDREPEAT;

    IF (list A is empty)
        THEN < no conflict table split has occurred. >
            CALL SEAM_FLAG_UPDATE;
        ELSE < conflict table has split. >
            Divide conflict table into two conflict tables -- aircraft on
               list B and pairs on list C form first table, while remaining
               aircraft and pairs on list A form second table;
            CALL SEAM_FLAG_UPDATE;   < for first conflict table >
            CALL SEAM_FLAG_UPDATE;   < for second conflict table >

END test_for_conflict_table_split;
```

```
--------------------------------------------------------------------------------
PROCESS test_for_conflict_table_split;

    PTR PACS;

    Create a linked list (list A) of all pair records remaining in conflict table;
    Place pointer to first conflict table entry in list B;
    REPEAT UNTIL (list A is empty OR list B is empty);

        PACS = next aircraft ID from list B;   < subject aircraft >
        LOOP;   < Repeat for each pair in list A. >
            IF (PACS EQ PREC.ac1.PAC OR PACS EQ PREC.ac2.PAC for this pair)
                THEN Remove this pair from list A;
                     Add this pair to list C;
                     Add PAC of other aircraft in this pair to bottom of list B;
        EXITIF (all pairs in list A examined);
        ENDLOOP;

    ENDREPEAT;

    IF (list A is empty)
        THEN CALL SEAM_FLAG_UPDATE IN (Central track store)
                                   INOUT (Conflict table);
        ELSE Divide conflict table into two conflict tables -- aircraft on
                list B and pairs on list C form first table, while remaining
                aircraft and pairs on list A form second table;
            CALL SEAM_FLAG_UPDATE IN (Central track store)
                                   INOUT (First conflict table);
            CALL SEAM_FLAG_UPDATE IN (Central track store)
                                   INOUT (Second conflict table);

END test_for_conflict_table_split;
```

```
-------------------------------------------------------------------------------------
ROUTINE SEAM_FLAG_UPDATE
   IN (Central track store)
   INOUT (Conflict table);

     < This routine determines the setting of the seam flag for a conflict table. >

     CLEAR seam flag in conflict table;

     LOOP:   < Repeat for each aircraft in conflict table. >

          IF (bit set in GEOG field for any connected site)
               THEN SET seam flag;

     EXITIF (seam flag set OR all aircraft in conflict table have been selected);
     ENDLOOP;

END SEAM_FLAG_UPDATE;
```

```
--------------------------------------------------------------------------------
ROUTINE SEAM_FLAG_UPDATE
   IN (Central track store)
   INOUT (Conflict table);

     CLEAR CTHEAD.SEAM;

     LOOP:   < Repeat for each aircraft in conflict table. >

          IF (SVECT.GEOG NE SYSTEM.OWNID)
               THEN SET CTHEAD.SEAM;

     EXITIF (CTHEAD.SEAM EQ TRUE OR all aircraft in conflict table selected);
     ENDLOOP;

END SEAM_FLAG_UPDATE;
```

```
--------------------------------------------------------------------------------
ROUTINE UPDATE_SECTOR_ID
   INOUT (State vectors, conflict table, and pair record for a conflict):

     < This routine updates the sector ID in a pair record. >

     IF (either aircraft is remote)
          THEN sector ID = sector of non-remote aircraft;
     ELSEIF (both aircraft in same sector)
          THEN sector ID = common aircraft sector;
     ELSEIF (aircraft are less than three sectors apart)
          THEN sector ID = sector swept first by radar beam;
     OTHERWISE sector ID = aircraft sector swept last by radar beam;

END UPDATE_SECTOR_ID;
```

```
--------------------------------------------------------------------------------
ROUTINE UPDATE_SECTOR_ID
   INOUT (ACID1, ACID2, conflict table, pair record);

     INT DELTA;

     IF (CTENTRY.REMFLG EQ $TRUE for either aircraft)
          THEN PREC.SECTID = SVECT.SVSID of non-remote aircraft;
     ELSEIF (SVECT1.SVSID EQ SVECT2.SVSID)
          THEN PREC.SECTID = SVECT1.SVSID;
     OTHERWISE DELTA = ABS(SVECT2.SVSID - SVECT1.SVSID);
               IF (DELTA LE NEAR1 OR
                   (DELTA GT HALFSEC AND DELTA LT NEAR2))
                    THEN PREC.SECTID = MIN(SVECT1.SVSID, SVECT2.SVSID);
                    ELSE PREC.SECTID = MAX(SVECT1.SVSID, SVECT2.SVSID);

END UPDATE_SECTOR_ID;
```

```
--------------------------------------------------------------------------------

TASK CONFLICT_PAIR_CLEANUP
   IN (Central track store, ID of current sector)
   INOUT (Linked list of conflict tables);

     < This task serves as a backup to the Resolution Deletion Task to ensure that
       conflicts are closed out in the proper manner.  It searches the linked list
       of conflict tables for a sector to find conflict pairs not processed on the
       current scan. >

     REPEAT WHILE (there are more conflict tables to be examined);

          Select next conflict table;
          REPEAT WHILE (there are more pair records to be examined);

               Select next pair record;

               IF (sector ID in pair record EQ current sector AND
                   this pair has not been processed on current scan)

                    THEN CALL CONFLICT_CLOSEOUT;
                         IF (okay to delete pair record)
                              THEN CALL PAIR_RECORD_DELETION;
                              ELSE CLEAR flag which indicates pair has been processed;
                                   Set pointer to list of potential domino conflict
                                      aircraft to null for both aircraft;

                    ELSE :   < do not process this pair. >

          ENDREPEAT;

     ENDREPEAT;

END CONFLICT_PAIR_CLEANUP;


------------------ CONFLICT PAIR CLEANUP TASK HIGH-LEVEL LOGIC --------------------
```

```
---------------------------------------------------------------------------------
TASK CONFLICT_PAIR_CLEANUP
   IN (Central track store, ID of current sector)
   INOUT (Linked list of conflict tables);

     BIT PRDELOK;

     REPEAT WHILE (there are more conflict tables to be examined);

          Select next conflict table;
          REPEAT WHILE (there are more pair records to be examined);

               Select next pair record;

               IF (PREC.SECTID EQ current sector AND
                   PREC.PWISP EQ $FALSE)

                    THEN CALL CONFLICT_CLOSEOUT
                                   IN (ACID1, ACID2, conflict table, pair record)
                                   OUT (PRDELOK);
                         IF (PRDELOK EQ $TRUE)
                              THEN CALL PAIR_RECORD_DELETION
                                             IN (ACID1, ACID2, pointer to pair record)
                                             INOUT (Conflict tables, central track store);
                              ELSE CLEAR PREC.PWISP;
                                 PREC.ac1.INTR = $NULL;
                                 PREC.ac2.INTR = $NULL;

                    ELSE ;  < do not process this pair. >

          ENDREPEAT;

     ENDREPEAT;

END CONFLICT_PAIR_CLEANUP;

------------------- CONFLICT PAIR CLEANUP TASK LOW-LEVEL LOGIC --------------------
```

```
-------------------------------------------------------------------------------
TASK STATE_VECTOR_DELETION
   IN (Deletion list, linked list of conflict tables)
   INOUT (Central track store);

     < This task is responsible for deleting the state vector of an aircraft which
       has left the ATARS/domino service area or which is no longer being
       adequately tracked. >

     REPEAT WHILE (there are more entries in the deletion list);

          Select next aircraft from deletion list;

          IF (there is no conflict table for this aircraft)
               THEN the state vector is to be deleted;
          ELSEIF (there are no conflicts involving this aircraft for which the
                     uplink of resolution advisory messages is still being attempted)
               THEN The state vector is to be deleted;
                    Create an entry in REMA or REMD for this aircraft;
          OTHERWISE the state vector is not to be deleted;

          IF (the state vector is to be deleted)
               THEN Erase CREPA or CREPD entry;
                    IF (this aircraft has an entry in CREPX)
                         THEN delete CREPX entry;
                    Unlink from sector thread;
                    Delete the state vector;

     ENDREPEAT;

END STATE_VECTOR_DELETION;

----------------- STATE VECTOR DELETION TASK HIGH-LEVEL LOGIC -------------------
```

```
--------------------------------------------------------------------------------

TASK STATE_VECTOR_DELETION
   IN (Deletion list, linked list of conflict tables)
   INOUT (Central track store);

     PTR ACID;
     BIT OKDEL;

     REPEAT WHILE (there are more entries in the deletion list);

          ACID = ID of next aircraft on deletion list;
          Access SVECT for this aircraft via ACID;

          IF (SVECT.CTPTR EQ $NULL)
               THEN SET OKDEL;
          ELSEIF (PREC.SEND not set in any pair records involving this aircraft)
               THEN SET OKDEL;
                    Create an entry in REMA or REMD for this aircraft;
          OTHERWISE CLEAR OKDEL;

          IF (OKDEL EQ $TRUE)
               THEN Erase CREFA or CREFD entry;
                    IF (SVECT.ATCREF NE $NULL)
                         THEN delete CREFX entry for this aircraft;
                    Unlink from sector thread;
                    Delete the state vector;

     ENDREPEAT;

END STATE_VECTOR_DELETION;
```

16. MESSAGE UPLINK PROCESSING

This section discusses the construction of the uplink messages to equipped aircraft. These messages are defined in the ATARS National Aviation Standard (Reference 9). All the messages discussed in this section are sent in the MA field of a DABS Comm-A message. Reference 9 provides complete details of these messages, their signal formats, and their coding.

16.1 Classes of ATARS Service

Reference 9 currently defines three classes of ATARS Service, denoted as Class 0, Class 1, and Class 2. This document contains the logic to service only these classes, although more may be defined in the future. The classes represent alternative levels of information. Messages from one of these classes are sent to an aircraft according to its airborne processing capabilities. The message sets are designed to minimize the processing required by the simplest user, and thus minimize his avionics cost; and to minimize the ATARS message load on the DABS channel by not sending certain information to those aircraft not equipped to process it.

Certain message types, namely the ATARS Resolution and the ATCRBS Track Block Messages, are common to all three classes of Service. These are described in the section for Class 0 service. Certain other types are also common to Classes 1 and 2. These are described in the section for Class 1 Service. A summary of all message types appears in Reference 9 and is repeated herein as Table 16-1.

16.1.1 ass 0 Service

Class 0 Service is intended for an aircraft with simple ATARS avionics. The avionics is assumed not to store traffic advisory data in a "track file," as described below for higher classes of service. The avionics may only display position data for one traffic advisory at a time, or it may be capable of displaying several traffic advisories. To allow for this limited capability, traffic advisories are ordered in decreasing order of "importance" so that the pilot will be shown the most urgent one(s). The following sections define the messages for Class 0 Service.

[illegible].1.1.1 ATARS Resolution Message

[illegible] message is the same for all classes of service. It [illegible] a 14-bit COL field, which is the column to be written

TABLE 16-1

MA SUBFIELD STRUCTURE OF ATARS MESSAGES

ATARS MESSAGE NAME	CLASS OF ATARS SERVICE	SUBFIELD		STRUCTURE OF MA
		Bit33	---------Bit40	Bit 41-----------------------------------Bit88
		(ADS1:4)	(ADS2:4)	
ATARS Resolution	0,1,2	=3	=0	(COL:14)(TRA:19)(***:12)(SIT:3)
Not Assigned	-	=3	=1-15	-
Six Advisories	0	=1	=0	(***:12)(***:12)(SIX:6)(SIX:6)(SIX:6)(SIX:6)
Three Advisories	0	=1	=1	(***:18)(***:18)(***:12)
Not Assigned	-	=1	=2	-
ATCRBS Track Block	0,1,2	=1	=3	Note: Defined in Ref. 11: 3.3.2.3.2 and is sent only to BCAS equipped aircraft.
Not Assigned	=1	=1	=4-6	-
Auxiliary Advisories	1,2	=1	=7	When RST=0 (TER:8)(OBT:13)(RST:1)(OBT:13)(OBT:13) When RST=1 (TER:8)(OBT:13)(RST:1)(RTD:26)
Own Plus Altitude Echo	2	=1	=8	(ODS:24)(AEC:24)
Own Plus Proximity	2	=1	=9	(ODS:24)(PDT:24)
Start/End Encounter	2	=1	=10	(PDT:24)(SED:24)
Dual Proximity	1,2	=1	=11	(PDT:24)(PDT:24)
Proximity Plus Altitude Echo	1,2	=1	=12	(PDT:24)(AEC:24)
Start Threat	2	=1	=13	(ODS:24)(STD:24)
Threat	1,2	=1	=14	(PDT:24)(THD:24)
Not Assigned	-	=1	=15	-

(***:12) can be either (TPA:12), (TEA:12) (OBA:12) OR (RAA:12), or a Null Advisory

(***:18) can be either (TPE:18), (AEE:18), (TAE:18) (OAE:18), OR (RAE:18), or a Null Advisory

Note: (XXX:N) denotes a subfield designated "XXX" which is assigned N bits.

into the aircraft RAR column designated by the SIT field. The COL field represents the composite of all resolutions the site is sending the aircraft for all conflict pairs. The site repeats the message each scan during the conflict, and sends this message again with COL containing all zeros, once at the end of the conflict to remove its advisories from the RAR. SIT corresponds to the ID of the site originating the advisories. It is normally set to own-ID. In the backup-master mode, it may contain the failed site's ID. When performing remote uplink, it contains the requesting site's ID.

This message contains the TRA subfield which describes the threat or proximate aircraft causing the resolution advisory. In the case of a multi-aircraft encounter, the most critical threat is used. A resolution advisory should always be caused by an aircraft whose Encounter List entry is a Threat type. However, on the scan when the final (zero) COL field is sent, the only traffic remaining may have entries with Proximity type.

The resolution message also contains a 12-bit subfield. This may contain another threat or proximity advisory, or a terrain, airspace, or obstacle advisory.

16.1.1.2 ATCRBS Track Block Message

This message is sent only to BCAS-equipped aircraft. Its generation (Section 8.1) is dependent upon BCAS logic, and is independent of any ATARS traffic and/or resolution advisories to BCAS for the same ATCRBS traffic. Up to eight such messages may be sent to each BCAS aircraft, depending upon the number of qualifying ATCRBS tracks. The message contains a SIT field as in the ATARS Resolution Message, and a track number to aid BCAS in associating messages with the same track on subsequent scans. The track data contains range, range rate, altitude, altitude rate, and bearing. ATARS tracked data is predicted ahead one scan to the time of transmission of the message.

16.1.1.3 Three Advisories Message

If no ATARS Resolution Message is constructed, this message is used to send up to three advisories. The first two advisories are contained in 18-bit subfields, and the third in a 12-bit subfield as described above for the ATARS Resolution Message. The 18-bit subfields may contain any of the following: a threat or proximity advisory, which is like that of the 12-bit subfield with additional data for range and fine altitude; terrain, airspace or obstacle advisories containing the same data as for the 12-bit subfield; or an altitude echo. If there are at least two traffic advisories, these will use the 18-bit subfields and any altitude echo in the PWILST will not be sent. If there are less than two traffic advisories, an altitude echo will always be sent in an 18-bit subfield, unless sending the altitude echo creates a need for an extra message.

16.1.1.4 Six Advisories Message

If more advisories remain to be sent after either the ATARS Resolution or the Three Advisories Messages are constructed, up to six additional advisories may be sent in this message. These are limited to fewer bits than those preceding but would correspond to less important advisories. Any remaining advisories after these six would not be sent to the aircraft this scan.

The first two advisories use the same 12-bit subfield described above. The remaining four advisories are contained in 6-bit subfields. These subfields may only contain proximity or threat advisories. Thus, if the proximity and threat advisories on the PWILST have not all been sent before reaching the 6-bit subfields, any terrain, airspace, or obstacle advisories will not be sent. This only happens when there are at least five traffic advisories, or at least four traffic advisories and a resolution advisory.

16.1.2 Class 1 Service

Class 1 Service will support a graphic display and is intended for aircraft capable of displaying larger quantities of data than the Class 0 aircraft. In most cases, more messages will be sent to such aircraft to uplink the same number of advisories. The ATARS Resolution Message and the ATCRBS Track Block Message are as described in Sections 16.1.1.1 and 16.1.1.2. However, any traffic advisory data contained in the ATARS Resolution Message is repeated using the messages below.

16.1.2.1 Dual Proximity Message

This message contains position data for two proximity advisories. Each advisory is sent in a Position Data Subfield, which contains clock and fine bearing, altitude zone and fine altitude, range, the heading of the traffic, and the control state and ATARS equipage of the traffic. A first-time transmitted bit is sent to denote new traffic.

16.1.2.2 Threat Message

This message contains one threat advisory. It consists of a Position Data Subfield as above, plus a Threat Data Subfield containing an altitude extension, fine heading, horizontal miss distance, turn type, vertical speed of the threat, and an indication as to whether the threat is causing a resolution advisory to be sent to own aircraft.

16.1.2.3 Proximity Plus Altitude Echo Message

This message combines one proximity advisory with altitude echo data. It contains the last (i.e. least important) proximity advisory. If no single proximity advisory remains after all Dual Proximity Messages are built (if any), and an altitude echo is required, a null Position Data Subfield will be inserted to complete this message.

16.1.2.4 Auxiliary Advisories Message

This message contains terrain, obstacle, and airspace advisories. Its format contains a terrain warning (if needed), an obstacle advisory (if needed), and either an airspace advisory or up to two additional obstacle advisories. Both its obstacle and airspace advisories include specific identification data not provided in the Class 0 formats. A first-time transmitted bit is sent to denote each new advisory.

16.1.3 Class 2 Service

Class 2 Service is intended for aircraft with sophisticated avionics capable of tracking traffic from scan to scan. Class 2 messages include the entire set of Class 1 messages plus additional types intended to aid such avionics. These additional messages are described below.

16.1.3.1 Start/End Encounter Message

This message helps the avionics start a track by assigning a track number to the traffic. The message contains a Position Data Subfield and a Start/End Subfield. The latter indicates whether to begin or end the track, assigns the track number, and contains the groundspeed, climb performance, and abbreviated identification data for the traffic. Up to eight unique track numbers may be assigned to tracks for an aircraft's traffic advisories. The End Encounter Message is normally sent at the conclusion of traffic advisory status. However, if more than eight tracks qualify simultaneously, ATARS sends a Start

Encounter Message when a new track is chosen to replace an old one with the same number, and this implies the end of the old track.

16.1.3.2 Own Plus Proximity Message

This message combines a single proximity advisory with own-aircraft data. The own-aircraft data allows the avionics to more accurately relate advisory data to own aircraft's heading, speed and turn rate. It also confirms the class of ATARS service in use. A bit indicates the initiation or handoff of ATARS service, to indicate to the avionics possible track number discontinuities if sensors change ATARS responsibility during an encounter.

16.1.3.3 Own Plus Altitude Echo Message

This message combines own-aircraft data with altitude echo data and is uplinked when both of these types are required.

16.1.3.4 Start Threat Message

This message combines own-aircraft data with a Start Threat Data Subfield. The latter is the same as a Start Encounter Message, except the Start/End bit is replaced by a bit indicating whether the threat is a new track or is an upgrade of an existing proximity advisory to a threat.

16.2 Data Link Message Construction Task

The Data Link Message Construction (DLMC) Task assembles messages for uplink to each equipped aircraft in the ATARS service area. The following sections describe the functions performed by this task.

16.2.1 Ranking PWILST Entries

The uplink messages are eventually constructed from the entries on the aircraft's PWILST. These entries are created in the order that pairs on the Encounter List are processed. However, the desired order of their uplink is determined by factors independent of this original ordering. The ranking procedure reorders the list, enabling the subsequent procedures to travel through the list and assemble messages in the desired order.

Assuming all types are present, the Entry Ranking Process produces the ordering of entries shown in Table 16-2. The groups numbered 2, 4, and 9 are selected using only the TYPE of the entry. For all TA_PROX or TA_THREAT entries, the Entry Ranking Process calculates the current value of each entry's RANKTYP field according to the criteria shown in the "Meaning" column of Table 16-2.

Within each group containing traffic advisory entries, the entries are ordered according to additional rank data. The formats of these fields are also shown in Table 16-2. The Traffic Advisory Task (Section 9) computes the "tau" and "weighted range" fields each scan, using data from the encounter list entry. The DLMC Task assigns the RANKTYP. Both tau and weighted range are stored in two's complement form. These fields are concatenated and then interpreted as a single unsigned integer. This interpretation causes entries with the smallest positive values of tau and range (i.e. shortest tau and closest range) to receive the highest ranking within each group. Both tasks update their respective data each scan that the entry is refreshed. Within groups 1 and 3 (separately), the ranking procedure orders entries by tau and weighted range. Within groups 5 through 8 (separately), this procedure orders proximities with mode C altitude reports ahead of non-mode C proximities, and within each group, orders entries by weighted range. Within groups 2, 4, and 10, the order is immaterial. Group 9 can have only one entry. This ranking orders the entries by importance, and avoids problems in class 2 message construction, in which advisories in a Start/End Encounter Message use a full message, while proximities may be paired together or paired with ALEC or Own-aircraft data. The only adverse impact of this ranking for Class 0 or 1 users is minor: a new proximity advisory may precede another for closer traffic on one scan.

There are no OWN entries on the PWILST, as all the logic to generate Own-aircraft data subfields is handled locally within the DLMC Task. Also, group 10 entries (END Threat or Prox) are only kept for class 2 ATARS users.

16.2.2 Altitude Echo

Altitude Echo (ALEC) PWILST entries are created for any of several reasons:

TABLE 16-2

RANKING ENTRIES ON THE PWILST

GROUP	ENTRY TYPE	RANKTYP FIELD	MEANING OF RANKTYP FIELD
1.	TA_THREAT	1100	SVECT.CTPTR non-null
2.	ATCRBS_TB	- -	
3.	TA_THREAT	1000	SVECT.CTPTR null and TA_THREAT.END not set
4.	TERRAIN AIRSPACE, OBSTACLE	- -	
5.	TA_PROX	0101	SVECT. MCFLG set and TA_PROX.OLD_TYPE = "none"
6.	TA_PROX	0100	SVECT.MCFLG not set and TA_PROX.OLD_TYPE = "none"
7.	TA_PROX	0011	SVECT.MCFLG set and TA_PROX.OLD_TYPE not "none" and TA_PROX.END not set
8.	TA_PROX	0010	SVECT.MCFLG not set and TA_PROX.OLD_TYPE not "none" and TA_PROX.END not set
9.	ALEC	- -	
10.	TA_PROX, TA_THREAT	0000	END set

TABLE 16-2
(Concluded)

-Tau Field

Set to zero for TA_PROX types.
For TA_THREAT types, two's complement of "Tau" (stored by Traffic Advisory Task)

-Weighted Range Field

Two's complement of weighted range (stored by Traffic Advisory Task)

Note: The -Tau and -Weighted Range fields are concatenated and interpreted as a single binary unsigned integer in the ranking process.

a. When a pilot sends an ALEC request (received in a DABS surveillance report), the Report Processing Task (Section 4.4) creates an ALEC entry.

b. When an equipped aircraft enters ATARS service, the DLMC Task notes that ALECT in the State Vector is uninitialized, and creates an ALEC entry.

c. When sufficient time has elapsed since ALECT, the time of the last ALEC message to the aircraft, the DLMC Task creates an ALEC entry.

d. When an ALEC uplink fails, the uplink delivery notice process in Non-surveillance Message Processing Task (Section 5.1) resets ALECT to an uninitialized value to force an immediate retry as in b. above.

As stated in Section 16.1, in periods of heavy traffic, the low priority assigned ALEC may cause this message not to be assigned a field if the user is ATARS Class 0, or to not be delivered if many uplinks are scheduled, for other user classes. In all cases, ALEC will be retried as soon as traffic becomes light enough.

16.2.3 Construction of Uplink Messages

After ordering PWILST entries and assigning track numbers, the DLMC Task constructs as many uplink message MA fields (see References 9, 10) as required to send ATARS advisories to each equipped aircraft. The pseudocode specifies the fields that are to be built within each message. The detailed coding of these fields is in Reference 9. For Class 1 and 2 users, every 48-bit MA field contains two 24-bit fields, as shown in Table 16-1. The pseudocode uses a local variable SUBFIELDNO to indicate whether the first or second of these in an uplink mesage is next to be filled. The ADS code is added to indicate the type of message and define the subfields that follow. All completed messages are sent as uplinks, as specified in Section 3.2.

16.2.4 Deleting PWILST Entries

The various types of PWILST entries are removed in different ways. Some types are sent only once; others are dropped when not refreshed; and some need an end message uplinked.

TA types (THREAT and PROX) contain an END item which is set by uplink delivery notice processing after they are successfully delivered. If an entry again qualifies for TA status, the Traffic Advisory Task resets END. If not, the DLMC Task Entry Ranking Process deletes these entries if the user is Class 0 or 1, as an end message is not sent to these users. If the user is Class 2, the DLMC Task builds an End Encounter Message and then immediately deletes the PWILST entry.

An ALEC entry is deleted by the DLMC Task immediately after the message is built.

Terrain, Airspace, and Obstacle types also contain an END item which is set by uplink delivery notice processing after successful delivery. The T/A/O Task resets END each scan that the alert continues. When the alert ends, END is not reset and the DLMC Task Entry Ranking Process deletes the entry.

ATCRBS Track Blocks likewise contain END. A single attempt to send an End Track Block is made for these types. The DLMC Task sets END when the ATCRBS Track Block Message is constructed. If the Traffic Advisory Task does not reset END on the next scan, the DLMC Task sends an End Track Block and deletes the PWILST entry.

16.3 Pseudocode for Message Uplink Processing

The following comments will clarify the implementation of the Data Link Message Construction Task. The pseudocode repeatedly refers to PWILST entries of a particular TYPE. While TYPE is not a field in the entry, the definitions of PWILST entries in Section 3 pseudocode identify the TYPE for each.

The Entry Ranking Process makes several passes through an aircraft's PWILST. These passes include the RANKTYP assignment and reordering discussed in Section 16.2.1, and the assignment of track numbers. It is suggested that temporary lists be kept of unused track numbers and of ATCRBS Track Block track numbers.

The various message generation processes primarily look down an aircraft's PWILST to find the next entry which has not been marked SENT. An exception is the Auxiliary Advisories Message Generation Process, which builds the Auxiliary Advisories Message from all T/A/O entries on the PWILST and marks them all SENT.

As uplink messages are built, the pseudocode assigns an ADS code. These are shown as the appropriate ATARS Message name.

See Table 16-1 for the corresponding numerical values. The priority bit included with each message is not uplinked to the aircraft, and thus is not contained in the formats shown in Table 16-1. This bit is used by the DABS sensor in its message scheduling (Reference 1). The phrase "move to uplink buffer" means "build the complete message format as shown in References 1 and 8". If SVECT.REMRAR is set to a positive value, the uplink messages should be routed to the remote sensor indicated. In all other cases, the local sensor performs the uplink.

As each message is moved to the uplink buffer, a copy is linked to a list kept for the aircraft. This list, which is discarded and created anew each scan, is pointed to by SVECT.UPMES.

PSEUDOCODE TABLE OF CONTENTS

PAGE

```
--------------------------------------------------------------------------------

STRUCTURE DLMCPARM
   GROUP change_thresholds
     INT ALTCTIM                          <time to generate new ALTC entry>
     FLT OWN_DELTA_HDG                    <heading change requiring Own Message>
ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

AD-A104 148 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-2 FAA-RD-81-45-2 NL

6 OF 7
AD
A104148

```
--------------------------------------------------------------------------------
STRUCTURE DLMCVBL
   GROUP miscellaneous
     BIT OWN_REQD                        <Own Message is required>
     INT PROXNO                          <counter for first/second traffic
                                                 advisory in message>

     INT SUBFIELDNO                      <counter for subfield in message>
     BIT RESSENT                         <Resolution Message sent>
ENDSTRUCTURE:
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------
TASK DATA_LINK_MESSAGE_CONSTRUCTION
   IN (Sector list of aircraft, state vectors)
   OUT (messages to uplink buffer)
   INOUT (PWILST's);

     <Merge all uplinks to aircraft into message structure.>

     REPEAT WHILE (more aircraft on list);
       Select next aircraft;
       IF (ATARS equipped AND in ATARS service)
          THEN PERFORM altitude_echo_test; <determine if alt. echo message required>
               PERFORM entry_ranking; <reorder entries on PWILST>
               IF (aircraft is in a conflict table AND own site giving resolution)
                    THEN PERFORM resolution_message_generation;

            <build remaining messages according to class of ATARS service>
               IF (TA_class EQ 0)
                    THEN PERFORM class_0_DLMC;
               ELSEIF (TA_Class EQ 1)
                    THEN PERFORM class_1_DLMC;
               OTHERWISE PERFORM class_2_DLMC;

               Link set of uplink messages to state vector;

     ENDREPEAT;

END DATA_LINK_MESSAGE_CONSTRUCTION;
```

-------------- DATA LINK MESSAGE CONSTRUCTION TASK HIGH-LEVEL LOGIC ---------------

```
TASK DATA_LINK_MESSAGE_CONSTRUCTION
   IN (Sector list of aircraft, state vectors)
   OUT (messages to uplink buffer)
   INOUT (PWILST's);

     <merge all uplinks to aircraft into message structure>

     REPEAT WHILE (more aircraft on sector list);
       Select next aircraft;
       IF (SVECT.ATSEQ NE SUMEQ AND SVECT.ATSS EQ $TRUE)
          THEN PERFORM altitude_echo_test; <determine if alt. echo message required>
               PERFORM entry_ranking; <reorder entries on PWILST>
               CLEAR RESSENT;
               IF (SVECT.CTPTR not null AND pair rec found for AC with PREC.SEND set)
                    THEN PERFORM resolution_message_generation;

            <build remaining messages according to class of ATARS service>
               IF (SVECT.ACLASS EQ $CL0)
                    THEN PERFORM class_0_DLMC;
               ELSEIF (SVECT.ACLASS EQ $CL1)
                    THEN PERFORM class_1_DLMC;
               OTHERWISE PERFORM class_2_DLMC;

               Link SVECT.UPMES to set of uplink messages;

     ENDREPEAT;

END DATA_LINK_MESSAGE_CONSTRUCTION;
```

```
------------------------------------------------------------------------------------

PROCESS altitude_echo_test;

    <See if required to generate ALEC entry, even if no
     pilot request. If no ALEC sent recently, send one now.>

    IF (no ALEC entry on PWILST)
         THEN IF (ALECT uninitialized OR sufficient time since ALECT)
                   THEN Create ALEC entry and link to bottom of PWILST;
                        Update ALECT with current time;

END altitude_echo_test;
```

```
-------------------------------------------------------------------------------
PROCESS altitude_echo_test;

     <see if required to generate ALEC entry, even if no
         pilot request. If no ALEC sent recently, send one now>

     IF (no ALEC entry on PWILST)
          THEN IF (SVECT.ALECT uninitialized OR SYSVAR.CTIME-SVECT.ALECT GT ALECTIM)
                    THEN Create ALEC entry and link to bottom of PWILST;
                         SVECT.ALECT=SYSVAR.CTIME;

END altitude_echo_test;
```

```
------------------------------------------------------------------------------------
PROCESS entry_ranking:
     REPEAT WHILE (PWILST contains more Prox or Threat entries);
          IF (TA_class LT 2 AND END set)
               THEN Delete entry; <don't send END msg to class 0 or 1>
               ELSE Determine and store rank type;
     ENDREPEAT:
     REPEAT WHILE (PWILST contains more entries);
          Sort all entries into descending rank order;
          CLEAR SENT flag for each entry;
     ENDREPEAT:

     IF (TA_class EQ 0)
          THEN Delete all prox or threat entries after first 9;
          ELSE Delete all prox or threat entries after first 8;
               REPEAT WHILE (any remaining have uninitialized Track_no);
                    Assign lowest unused Track_no;
               ENDREPEAT:
     REPEAT WHILE (any Terrain, Airspace, Obstacle entries);
          IF (END set)   <entry not updated this scan>
               THEN Delete entry;   <don't send END msg for these types>
     ENDREPEAT:
     REPEAT WHILE (any ATCRBS Track Block entries);
          IF (ATCRBS_Track_no uninitialized)
               THEN Assign lowest unused ATCRBS_Track_no;
                    IF (this ATCRBS_Track_no GT 7)
                         THEN IF (any ATCRBS_TB entry found with END status AND next
                                   entry does not have same ATCRBS_Track_no)
                                   THEN Move subject entry after one with END status;
                                        Assign subject entry same ATCRBS_Track_no
                                           as entry with END status;
                                   ELSE Delete subject entry; <too many Track Blocks>
     ENDREPEAT:

END entry_ranking;

-------------- DATA LINK MESSAGE CONSTRUCTION TASK HIGH-LEVEL LOGIC ---------------
```

```
--------------------------------------------------------------------------------

PROCESS entry_ranking;
     REPEAT WHILE (PWILST contains more TA_PROX or TA_THREAT entries);
          IF (SVECT.ACLASS NE $CL2 AND END set)
               THEN Delete entry; <don't send END msg to class 0 or 1>
               ELSE Determine and store RANKTYP; <see Table 16-2 for details>
     ENDREPEAT;
     REPEAT WHILE (PWILST contains more entries);
          Sort all entries into descending rank order;
          CLEAR SENT flag for each entry;
     ENDREPEAT;


     IF (SVECT.ACLASS EQ $CL0)
          THEN Delete all prox or threat entries after first 9;
          ELSE Delete all prox or threat entries after first 8;
               REPEAT WHILE (any remaining have uninitialized TRACK_NO):
                    TRACK_NO = lowest unused value;
               ENDREPEAT;
     REPEAT WHILE (any TERRAIN, AIRSPACE, OBSTACLE entries):
          IF (END set)   <entry not updated this scan>
               THEN Delete entry;   <don't send END msg for these types>
     ENDREPEAT;
     REPEAT WHILE (any ATCRBS_TB entries);
          IF (ATCRBS_TRACK_NO uninitialized)
               THEN ATCRBS_TRACK_NO=lowest unused value;
                    IF (this ATCRBS_TRACK_NO GT 7)
                         THEN IF (any ATCRBS_TB entry found with END set AND next
                                   entry does not have same ATCRBS_TRACK_NO)
                                   THEN Move subject entry after one with END set;
                                        Assign subject entry same ATCRBS_TRACK_NO
                                           as entry with END set;
                                   ELSE Delete subject entry;
     ENDREPEAT;

END entry_ranking;

-------------- DATA LINK MESSAGE CONSTRUCTION TASK LOW-LEVEL LOGIC ---------------
```

```
--------------------------------------------------------------------------------
PROCESS resolution_message_generation;

    <Generate Resolution Advisory and other fields in Resolution message.>

     Build COL field from own site's resolution(s) for AC;
     Set SIT field to indicate own site;
     Select highest ranking traffic advisory entry;
     Build TRA field;  <null field if no entry found>
     SET entry SENT;
     PERFORM Subfield_12_bit_generation;
     ADS=Resolution;
     SET Priority bit in message;
     Move to uplink buffer;

END resolution_message_generation;
```

```
-----------------------------------------------------------------------------------
PROCESS resolution_message_generation;

    <generate Resolution Advisory and other fields in Resolution message>

     COL field=zeroes;
     REPEAT UNTIL (all pair records for AC processed);
          Select next PREC;
          IF (PREC.SEND EQ $TRUE)
               THEN COL=logical OR of COL field with PREC.PHMAN,PVMAN for AC;
                      <both $NORES, $NULLRES treated as zeroes>
     ENDREPEAT;
     IF (SYSCT.CENTR set)
          THEN SIT field=SYSVAR.FAILED;<send failed site's ID in Backup-Master mode>
          ELSE SIT field=SYSTEM.OWNID;
     Select highest ranking TA_THREAT or TA_PROX entry;
     Build TRA field;<null field if no such entry found>
     SET entry SENT;
     PERFORM Subfield_12_bit_generation;
     ADS=Resolution;
     SET Priority bit in message;
     Move to uplink buffer;
     SET RESSENT;

END resolution_message_generation;
```

```
--------------------------------------------------------------------------------

PROCESS class_0_DLMC;

                <Generate any msgs required for class 0 ATARS user.>

    REPEAT WHILE (PWILST contains more ATCRBS Track Block entries);
        Select next ATCRBS Track Block entry;
        PERFORM ATCRBS_track_block_uplink;
    ENDREPEAT;

    IF (Resolution message was not sent AND any entries on PWILST not yet sent)
        THEN PERFORM three_advisories_message_generation;
    IF (more PWILST entries not sent)
        THEN PERFORM six_advisories_message_generation;

END class_0_DLMC;
```

```
--------------------------------------------------------------------------------

PROCESS class_0_DLMC;

                    <generate any msgs required for class 0 ATARS user>

     REPEAT WHILE (PWILST contains more ATCRBS_TB entries);
          Select next ATCRBS_TB entry;
          PERFORM ATCRBS_track_block_uplink;
     ENDREPEAT;

     IF (RESSENT EQ $FALSE AND any entries on PWILST with SENT EQ $FALSE)
          THEN PERFORM three_advisories_message_generation;
     IF (more PWILST entries with SENT EQ $FALSE)
          THEN PERFORM six_advisories_message_generation;

END class_0_DLMC;
```

```
-------------------------------------------------------------------------------------

PROCESS class_1_DLMC;    <Generate msgs required for class 1 ATARS user.>

     CLEAR SENT flags on threat entries;
     <if Resolution message sent, need separate threat msgs for class_1 service>
     SUBFIELDNO=1;  <keep track of first or second subfield in msg>
     REPEAT WHILE (more PWILST entries not sent);
          Select next entry not sent;
          IF (TYPE=Threat)
               THEN build position data subfield, PTAT, threat data subfield;
                    ADS=Threat;
                    SET Priority bit in message;
                    Move to uplink buffer;
                    SET entry SENT;
          ELSEIF (Type = ATCRBS Track Block)
               THEN PERFORM ATCRBS_track_block_uplink;
          ELSEIF (Type = Terrain OR Airspace OR Obstacle)
               THEN PERFORM auxiliary_advisories_message_generation;
                    <send all T/A/O entries>
          ELSEIF (Type = Prox)
               THEN build position data subfield, PTAT; <TYPE=Prox>
                    SET entry SENT;
                    IF (SUBFIELDNO=2)
                         THEN Combine with saved subfield;
                              ADS=Dual Proximity;
                              CLEAR Priority bit in message;
                              Move to uplink buffer;
                              SUBFIELDNO=1;
                         ELSE SUBFIELDNO=2;
                              Save first subfield;
          OTHERWISE;  <don't process ALEC entry yet>
     ENDREPEAT;
     PERFORM class_1_altitude_echo_generation;

END class_1_DLMC;
```

```
--------------------------------------------------------------------------------
PROCESS class_1_DLMC;     <generate msgs required for class 1 ATARS user>

     CLEAR SENT flags on threat entries;
     <if Resolution message sent, need separate threat msgs for class_1 service>
     SUBFIELDNO=1;  <keep track of first or second subfield in msg>
     REPEAT WHILE (more PWILST entries with SENT EQ $FALSE):
          Select next such entry;
          IF (TYPE=TA_THREAT)
               THEN build position data subfield, PTAT, threat data subfield;
                    ADS=Threat;
                    SET Priority bit in message;
                    Move to uplink buffer;
                    SET entry SENT;
          ELSEIF (TYPE = ATCRBS_TB)
               THEN PERFORM ATCRBS_track_block_uplink;
          ELSEIF (TYPE = TERRAIN OR AIRSPACE OR OBSTACLE)
               THEN PERFORM auxiliary_advisories_message_generation;
                   <send all T/A/O entries>
          ELSEIF (TYPE = PROX)
               THEN build position data subfield, PTAT;
                    SET entry SENT;
                    IF (SUBFIELDNO=2)
                         THEN Combine with saved subfield;
                              ADS=Dual Proximity;
                              CLEAR Priority bit in message;
                              Move to uplink buffer;
                              SUBFIELDNO=1;
                         ELSE SUBFIELDNO=2;
                              Save first subfield;
          OTHERWISE:  <don't process ALEC entry yet>
     ENDREPEAT;
     PERFORM class_1_altitude_echo_generation;

END class_1_DLMC;

-------------- DATA LINK MESSAGE CONSTRUCTION TASK LOW-LEVEL LOGIC ----------------
```

```
--------------------------------------------------------------------------------
PROCESS class_2_DLMC:    <Generate messages for class 2 ATARS user.>

    PERFORM own_message_requirement_test;
    CLEAR SENT flags on threats;  <need separate msgs for class 2>
    PROXNO=1;
    REPEAT WHILE (more entries not sent);
        Select next entry not sent;
        IF (TYPE=Threat) AND (END not set)
            THEN PERFORM class_2_threat_generation;
            <may create own data, start threat subfields>
        ELSEIF (Type = ATCRBS Track Block)
            THEN PERFORM ATCRBS_track_block_uplink;
        ELSEIF (Type = Terrain OR Airspace OR Obstacle)
            THEN PERFORM auxiliary_advisories_message_generation;
                <send all T/A/O entries>
        ELSEIF ((Type = Prox or Threat) AND (OLD_TYPE=none OR END set))
                    <start or end prox or end threat entry>
            THEN Build position data subfield, PTAT, start/end subfield;
                ADS=Start/End Encounter;
                CLEAR Priority bit in message;
                Move to uplink buffer;
                IF (OLD_TYPE=none)  <start prox type>
                    THEN SET entry SENT;
                    ELSE Delete entry;    <end sent>
        ELSEIF (Type = Prox)
            THEN PERFORM continuing_prox_classification;
        OTHERWISE;  <don't process ALEC entry yet>
    ENDREPEAT;
    IF (Own message required and not sent yet)
        THEN Build own data subfield;
            Update own msg time and data; <OWNT, OWNHDG, OWNTRN>
            CLEAR Own message required indication;
            PERFORM class_2_altitude_echo_generation;
    Delete any ALEC entry;
END class_2_DLMC:
-------------- DATA LINK MESSAGE CONSTRUCTION TASK HIGH-LEVEL LOGIC --------------
```

```
--------------------------------------------------------------------------------

PROCESS class_2_DLMC;    <generate messages for class 2 ATARS user>

     PERFORM own_message_requirement_test;
     CLEAR SENT flags on threats;  <need separate msgs for class 2>
     PROXNO=1;
     REPEAT WHILE (more entries with SENT EQ $FALSE);
          Select next such entry;
          IF (TYPE=THREAT) AND (TA_THREAT.END EQ $FALSE)
               THEN PERFORM class_2_threat_generation;
          ELSEIF (TYPE = ATCRBS_TB)
               THEN PERFORM ATCRBS_track_block_uplink;
          ELSEIF (TYPE = TERRAIN OR AIRSPACE OR OBSTACLE)
               THEN PERFORM auxiliary_advisories_message_generation;
                   <send all T/A/O entries>
          ELSEIF ((TYPE = PROX or THREAT) AND (OLD_TYPE=none OR END set))
                       <start or end prox or end threat entry>
               THEN Build position data subfield, FTAT, start/end subfield;
                    ADS=Start/End Encounter;
                    CLEAR Priority bit in message;
                    Move to uplink buffer;
                    IF (TA_PROX.OLD_TYPE=none)  <start prox type>
                         THEN SET TA_PROX.SENT;
                         ELSE Delete entry;    <end sent>
          ELSEIF (TYPE = PROX)
               THEN PERFORM continuing_prox_classification;
          OTHERWISE;  <don't process ALEC entry yet>
     ENDREPEAT;
     IF (OWN_REQD EQ $TRUE)
          THEN Build own data subfield;
               SVECT.OWNT=SYSVAR.CTIME;
               Update SVECT.OWNHDG, SVECT.OWNTRN;
               CLEAR OWN_REQD;
               PERFORM class_2_altitude_echo_generation;
     Delete any ALEC entry;
END class_2_DLMC;
-------------- DATA LINK MESSAGE CONSTRUCTION TASK LOW-LEVEL LOGIC ----------------
```

```
----------------------------------------------------------------------------------------

PROCESS ATCRBS_track_block_uplink:

     Select next ATCRBS Track Block entry;
     ADS=ATCRBS Track Block:
     SET Priority bit in message;
     SET entry SENT;
     Move to uplink buffer;
     IF (END field set)
          THEN delete PWILST entry;
                  <only make one try to send END msg for this type>
          ELSE SET END field;
                <if Detect task doesn't update this entry next scan,
                 the END field signals need for an END Track Block msg>

END ATCRBS_track_block_uplink;
```

```
PROCESS ATCRBS_track_block_uplink;

     Select next ATCRBS_TB entry;
     ADS=ATCRBS Track Block;
     SET Priority bit in message;
     SET ATCRBS_TB.SENT;
     Move to uplink buffer;
     IF (ATCRBS_TB.END EQ TRUE)
          THEN delete ATCRBS_TB entry;
                  <only make one try to send END msg for this type>
          ELSE SET ATCRBS_TB.END;
                <if Detect task doesn't update this entry next scan,
                 the END field signals need for an END Track Block msg>

END ATCRBS_track_block_uplink;
```

```
--------------------------------------------------------------------------------
PROCESS auxiliary_advisories_message_generation;

   <Search for T/A/O entries and build Auxiliary Advisories message.>

     IF (Terrain advisory entry on PWILST)
          THEN Build terrain advisory subfield;
               SET entry SENT;
          ELSE Build null terrain advisory subfield;
     PERFORM obstruction_subfield_generation;
     IF (Restricted Airspace entry found)
          THEN SET RST bit subfield;
               Build Restricted airspace advisory subfield;
               SET entry SENT;
          ELSE CLEAR RST bit subfield;
               PERFORM obstruction_subfield_generation;
               PERFORM obstruction_subfield_generation; <yes, two times>
             <message format includes either one Airspace subfield
                   or else two more obstruction subfields>
     ADS=Auxiliary Advisories;
     SET Priority bit in message;
     Move to uplink buffer;

END auxiliary_advisories_message_generation;
```

```
--------------------------------------------------------------------------------

PROCESS auxiliary_advisories_message_generation;

   <search for T/A/O entries and build Auxiliary Advisories message>

     IF (TERRAIN entry on PWILST)
          THEN Build terrain advisory subfield;
               SET TERRAIN.SENT;
          ELSE Build null terrain advisory subfield;
     PERFORM obstruction_subfield_generation;
     IF (AIRSPACE entry found)
          THEN SET RST bit subfield;
               Build Restricted airspace advisory subfield;
               SET AIRSPACE.SENT;
          ELSE CLEAR RST bit subfield;
               PERFORM obstruction_subfield_generation;
               PERFORM obstruction_subfield_generation; <yes, two times>
             <message format includes either one Airspace subfield
                   or else two more obstruction subfields>
     ADS=Auxiliary Advisories;
     SET Priority bit in message;
     Move to uplink buffer;

END auxiliary_advisories_message_generation;
```

```
--------------------------------------------------------------------------------
PROCESS class_1_altitude_echo_generation;

  <Find most efficient way to send Altitude Echo message.>

    IF (SUBFIELDNO=2)
         THEN IF (ALEC entry on PWILST)
                   THEN Use data from ALEC entry;
                   ELSE Generate ALEC data;
                        Store current time in ALECT;
              Build altitude echo subfield;
              Combine with saved subfield;
              ADS=Proximity plus Altitude Echo;
              CLEAR Priority bit in message;
              Move to uplink buffer;
              Delete ALEC entry (if any);
    ELSEIF (ALEC entry on PWILST) <need another msg to send ALEC alone>
         THEN Build null Position subfield;
              Build altitude echo subfield;
              ADS=Proximity plus altitude echo;
              CLEAR Priority bit in message;
              Move to uplink buffer;
              Delete ALEC entry;
    OTHERWISE:   <no prox. or ALEC>

END class_1_altitude_echo_generation;
```

```
-------------------------------------------------------------------------------------

PROCESS class_1_altitude_echo_generation;

  <find most efficient way to send Altitude Echo message>

     IF (SUBFIELDNO=2)
          THEN IF (ALEC entry on PWILST)
                    THEN Use data from ALEC entry;
                    ELSE Generate ALEC.adv_data;
                         SVECT.ALECT=SYSVAR.CTIME;
               Build altitude echo subfield;
               Combine with saved subfield;
               ADS=Proximity plus Altitude Echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               Delete ALEC entry (if any);
     ELSEIF (ALEC entry on PWILST) <need another msg to send ALEC alone>
          THEN Build null Position subfield;
               Build altitude echo subfield;
               ADS=Proximity plus altitude echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               Delete ALEC entry;
     OTHERWISE:   <no prox. or ALEC>

END class_1_altitude_echo_generation;
```

```
-------------------------------------------------------------------------------------
PROCESS class_2_altitude_echo_generation;

                    <Try to combine Own and ALEC subfields.>

    IF (ALEC entry on PWILST and not sent)
        THEN Build altitude echo subfield;
             ADS=Own Plus Altitude Echo;
    ELSEIF (no ALEC entry on PWILST)
        THEN Generate ALEC data;
             Store current time in ALECT;
             Build altitude echo subfield;
             ADS=Own Plus Altitude Echo;
    OTHERWISE Build null Position subfield; <ALEC already sent combined with Prox>
             ADS=Own Plus Prox;
    CLEAR Priority bit in message;
    Move to uplink buffer;

END class_2_altitude_echo_generation;
```

```
------------------------------------------------------------------------------------------

PROCESS class_2_altitude_echo_generation;

                        <try to combine Own and ALEC subfields>

    IF (ALEC entry on PWILST AND ALEC.SENT EQ $FALSE)
        THEN Build altitude echo subfield;
            ADS=Own Plus Altitude Echo;
    ELSEIF (no ALEC entry on PWILST)
        THEN Generate ALEC.adv_data;
            SVECT.ALECT=SYSVAR.CTIME;
            Build altitude echo subfield;
            ADS=Own Plus Altitude Echo;
    OTHERWISE Build null Position subfield; <ALEC already sent combined with Prox>
            ADS=Own Plus Prox;
    CLEAR Priority bit in message;
    Move to uplink buffer;

END class_2_altitude_echo_generation;
```

```
-----------------------------------------------------------------------------

PROCESS class_2_threat_generation;

          <Create one or two messages as required for Threat Advisory.>

    IF (entry has not already been sent as Threat type)
        THEN Build own data subfield; <required for new Threats>
            CLEAR Own message required indication;
               <don't also have to send separate Own message>
            Build Start Threat subfield;
            ADS=Start Threat;
            SET Priority bit in message;
            Move to uplink buffer;
    Build position data subfield, FTAT, threat data subfield;
    IF (threat causing resolution advisory)  <found in ranking process>
        THEN SET resolution advisory bit;
        ELSE CLEAR resolution advisory bit;
    ADS=Threat;
    SET Priority bit in message;
    Move to uplink buffer;
    SET entry SENT;

END class_2_threat_generation;
```

```
---------------------------------------------------------------------------------

PROCESS class_2_threat_generation;

          <Create one or two messages as required for Threat Advisory>

    IF (TA_THREAT.OLD_TYPE NE THREAT)
        THEN Build own data subfield; <required for new Threats>
            CLEAR OWN_REQD;
                <don't also have to send separate Own message>
            Build Start Threat subfield;
            ADS=Start Threat;
            SET Priority bit in message;
            Move to uplink buffer;
    Build position data subfield, STAT, threat data subfield;
    IF (TA_THREAT.RANKTYP=Threat causing resolution advisory)
        THEN SET resolution advisory bit;
        ELSE CLEAR resolution advisory bit;
    ADS=Threat;
    SET Priority bit in message;
    Move to uplink buffer;
    SET TA_THREAT.SENT;

END class_2_threat_generation;
```

```
--------------------------------------------------------------------------------
PROCESS continuing_prox_classification;
<Build msgs with Prox entries which are not start or end types (hence 'continuing').>

     Build position data subfield, PTAT;
     SET entry SENT;
     IF (PROXNO=2)
          THEN Combine with saved subfield;
               ADS=Dual Proximity;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               PROXNO=1;
     ELSEIF (more prox entries not yet sent) <not counting start,end types>
          THEN PROXNO=2;
               Save subfield;
     ELSEIF (ALEC entry on PWILST)
          THEN Build altitude echo subfield;
               ADS=Proximity Plus Altitude Echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               SET ALEC entry SENT;
     ELSEIF (Own message required and not yet sent)
          THEN build own data subfield;
               Update own message time and data; <OWNT, OWNHDG, OWNTRN>
               CLEAR Own message requirement indication;
               ADS=Own Plus Proximity;
               CLEAR Priority bit in message;
               Move to uplink buffer;
     OTHERWISE Create ALEC entry in PWILST;
               SET entry SENT;
               Store current time in ALECT;
               Build altitude echo subfield;
               ADS=Proximity Plus Altitude Echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
END continuing_prox_classification;
```

```
--------------------------------------------------------------------------------
PROCESS continuing_prox_classification;
     Build position data subfield, PTAT;
     SET TA_PROX.SENT;
     IF (PROXNO=2)
          THEN Combine with saved subfield;
               ADS=Dual Proximity;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               PROXNO=1;
     ELSEIF (more TA_PROX with (TA_PROX.SENT EQ $FALSE) AND
                   (TA_PROX.END EQ $FALSE) AND (TA_PROX.OLD_TYPE NE none))
          THEN PROXNO=2;
               Save subfield;
     ELSEIF (ALEC entry on PWILST)
          THEN Build altitude echo subfield;
               ADS=Proximity Plus Altitude Echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
               SET ALEC.SENT;
     ELSEIF (OWN_REQD EQ $TRUE)
          THEN build own data subfield;
               SVECT.OWNT=SYSVAR.CTIME;
               Update SVECT.OWNHDG, SVECT.OWNTRN;
               CLEAR OWN_REQD;
               ADS=Own Plus Proximity;
               CLEAR Priority bit in message;
               Move to uplink buffer;
     OTHERWISE Create ALEC entry in PWILST;
               SET ALEC.SENT;
               SVECT.ALECT=SYSVAR.CTIME;
               Build altitude echo subfield;
               ADS=Proximity Plus Altitude Echo;
               CLEAR Priority bit in message;
               Move to uplink buffer;
END continuing_prox_classification;
-------------- DATA LINK MESSAGE CONSTRUCTION TASK LOW-LEVEL LOGIC ----------------
```

```
--------------------------------------------------------------------------------
PROCESS obstruction_subfield_generation;

    IF (Obstacle entry found AND not SENT)
        THEN Build 13-bit Obstacle advisory subfield;
            SET entry SENT;
        ELSE Build null Obstacle advisory subfield;

END obstruction_subfield_generation;
```

```
--------------------------------------------------------------------------------------------

PROCESS obstruction_subfield_generation;

     IF (OBSTACLE entry found AND OBSTACLE.SENT EQ $FALSE)
          THEN Build 13-bit Obstacle advisory subfield;
               SET OBSTACLE.SENT;
          ELSE Build null Obstacle advisory subfield;

END obstruction_subfield_generation;
```

```
--------------------------------------------------------------------------------------

PROCESS own_message_requirement_test;

    <Use time, heading to see if own message required.>

     IF (OWNT uninitialized)    <AC just entered coverage OR last OWN uplink failed>
          THEN SET Own message requirement indication;
     ELSEIF (own ground track heading differs sufficiently
               from (prev. heading uplinked + turn rate uplinked * time since uplink))
          THEN SET Own message requirement indication;
     OTHERWISE CLEAR Own message requirement indication;

END own_message_requirement_test;
```

```
--------------------------------------------------------------------------------

PROCESS own_message_requirement_test;

    <Use time, heading to see if own message required>

     IF (SVECT.OWNT uninitialized)
          THEN SET OWN_REQD;
     ELSEIF (ABS(ARC TAN (SVECT.YD/SVECT.XD) - (SVECT.OWNHDG +
           SVECT.OWNTRN * (SYSVAR.CTIME-SVECT.OWNT))) GT OWN_DELTA_HDG)
          THEN SET OWN_REQD;
     OTHERWISE CLEAR OWN_REQD;

END own_message_requirement_test;
```

```
--------------------------------------------------------------------------------
PROCESS six_advisories_message_generation;

    REPEAT UNTIL (six fields generated);
        IF (first or second field)
            THEN PERFORM subfield_12_bit_generation;
            ELSE PERFORM subfield_6_bit_generation;
    ENDREPEAT;

    ADS=Six Advisories;
    IF (any subfield contains Threat, Terrain, Airspace, or Obstacle types)
        THEN SET Priority bit in message;
        ELSE CLEAR Priority bit in message;
    Move to uplink buffer;

END six_advisories_message_generation;
```

```
------------------------------------------------------------------------------------
PROCESS six_advisories_message_generation;

     REPEAT UNTIL (six fields generated);
          IF (first or second field)
               THEN PERFORM subfield_12_bit_generation;
               ELSE PERFORM subfield_6_bit_generation;
     ENDREPEAT;

     ADS=Six Advisories;
     IF (any subfield contains Threat, Terrain, Airspace, or Obstacle types)
          THEN SET Priority bit in message;
          ELSE CLEAR Priority bit in message;
     Move to uplink buffer;

END six_advisories_message_generation;
```

```
--------------------------------------------------------------------------------
PROCESS subfield_6_bit_generation:

     Find next Threat or Prox entry not sent;
     IF (entry found)
          THEN enter clock bearing in subfield indicator;
               Enter relative altitude zone;
               SET entry SENT;
          ELSE enter all zeros in subfield;  <no more Threat/Prox entries
                                              to fill the subfield>

END subfield_6_bit_generation;
```

```
--------------------------------------------------------------------------------

PROCESS subfield_6_bit_generation;

     Find next TA_THREAT or TA_PROX entry with SENT EQ $FALSE;
     IF (entry found)
          THEN enter clock bearing in subfield indicator;
               Enter relative altitude zone;
               SET entry SENT;
          ELSE enter all zeros in subfield;  <no more Threat/Prox entries
                                              to fill the subfield>

END subfield_6_bit_generation;
```

```
--------------------------------------------------------------------------------
PROCESS subfield_12_bit_generation;

     Find next PWILST entry not sent;
     IF (Type = Threat or Prox)
          THEN Set subfield indicator = clock bearing;
               Enter relative altitude zone, range bit, compass course;
               First time bit = FTAT;
               IF (Type=Threat)
                    THEN Threat/Prox bit =1;
                    ELSE Threat/Prox bit =0;
     ELSEIF (Type = Terrain)
          THEN Set subfield indicator = Terrain;
               Enter altitude relative to terrain data;
               First time bit = FTAT;
     ELSEIF (Type = Obstacle)
          THEN Set subfield indicator = Obstruction;
               Enter enter altitude relative to obstruction, clock bearing;
               First time bit = FTAT;
     ELSEIF (Type = Airspace)
          THEN Set subfield indicator = Airspace;
               Enter airspace type;
               First time bit = FTAT;
     OTHERWISE Set subfield indicator = Obstruction;  <no more entries>
               Set clock bearing = 0; <denotes null 12-bit subfield>
     SET entry SENT;

END subfield_12_bit_generation;
```

```
--------------------------------------------------------------------------------

PROCESS subfield_12_bit_generation;

    Find next PWILST entry with SENT EQ $FALSE;
    IF (TYPE = TA_THREAT or TA_PROX)
        THEN Set subfield indicator = clock bearing;
            Enter relative altitude zone, range bit, compass course;
            First time bit = FTAT;
            IF (TYPE=TA_THREAT)
                THEN Threat/Prox bit =1;
                ELSE Threat/Prox bit =0;
    ELSEIF (TYPE = TERRAIN)
        THEN Set subfield indicator = Terrain;
            Enter altitude relative to terrain data;
            First time bit = FTAT;
    ELSEIF (TYPE = OBSTACLE)
        THEN Set subfield indicator = Obstruction;
            Enter enter altitude relative to obstruction, clock bearing;
            First time bit = FTAT;
    ELSEIF (TYPE = AIRSPACE)
        THEN Set subfield indicator = Airspace;
            Enter airspace type;<see Reference 9 for coding>
            First time bit = FTAT;
    OTHERWISE Set subfield indicator = Obstruction;  <no more entries>
            Set clock bearing = 0; <denotes null 12-bit subfield>
    SET entry SENT;

END subfield_12_bit_generation;
```

```
---------------------------------------------------------------------------------------

PROCESS subfield_18_bit_generation;

    Find next PWILST entry not sent;
    IF (Type = Threat or Prox)
        THEN set subfield indicator = clock bearing;
            Enter alt. zone, rel. alt., range, compass course fields;
            First time bit = FTAT;
            IF (Type=threat)
                THEN Threat/Prox bit =1;
                ELSE Threat/Prox bit=0;
    ELSEIF (Type = Terrain AND not last non-ALEC entry) <if last, use 12-bit subf.>
        THEN Set subfield indicator = Terrain;
            Enter altitude relative to terrain data;
            First time bit =FTAT;
    ELSEIF (Type = Obstacle AND not last non-ALEC entry) <if last, use 12-bit subf.>
        THEN Set subfield indicator = Obstruction;
            Enter altitude relative to obstruction, clock bearing;
            First time bit = FTAT;
    ELSEIF (Type = Airspace AND not last non-ALEC entry) <if last, use 12-bit subf.>
        THEN Set subfield indicator = Airspace;
            Enter airspace type;
            First time bit = FTAT;
    ELSEIF ((Type = ALEC) OR ((no more entries OR exactly one T/A/O entry not sent)
                AND ALEC entry was not already sent))
        THEN Set subfield indicator = ALEC;
            Enter adj. altitude, mode C confidence, alt. echo data;
            IF (Type = ALEC exists on PWILST)
                THEN Delete ALEC entry;
                ELSE Store current time in ALECT;
    OTHERWISE Set subfield indicator = Obstruction; <no more PWILST entries>
            Set clock bearing=0; <denotes null 18-bit subfield>
    SET entry SENT;

END subfield_18_bit_generation;

-------------- DATA LINK MESSAGE CONSTRUCTION TASK HIGH-LEVEL LOGIC ---------------
```

```
-------------------------------------------------------------------------------

PROCESS subfield_18_bit_generation;

     Find next PWILST entry with SENT EQ $FALSE;
     IF (TYPE = TA_THREAT or TA_PROX)
          THEN set subfield indicator = clock bearing;
               Enter alt. zone, rel. alt., range, compass course fields;
               First time bit = FTAT;
               IF (TYPE=TA_THREAT)
                    THEN Threat/Prox bit =1;
                    ELSE Threat/Prox bit=0;
     ELSEIF (TYPE = TERRAIN AND not last non-ALEC entry)
          THEN Set subfield indicator = Terrain;
               Enter altitude relative to terrain data;
               First time bit =FTAT;
     ELSEIF (TYPE = OBSTACLE AND not last non-ALEC entry)
          THEN Set subfield indicator = Obstruction;
               Enter altitude relative to obstruction, clock bearing;
               First time bit = FTAT;
     ELSEIF (TYPE = AIRSPACE AND not last non-ALEC entry)
          THEN Set subfield indicator = Airspace;
               Enter airspace type;<see Reference 9>
               First time bit = FTAT;
     ELSEIF ((TYPE = ALEC) OR ((no more entries OR exactly one T/A/O entry not sent)
                AND no ALEC subfield already generated in this message)
          THEN Set subfield indicator = ALEC;
               Enter adj. altitude, mode C confidence, alt. echo data;
               IF (TYPE=ALEC)
                    THEN Delete ALEC entry;
                    ELSE SVECT.ALECT=SYSVAR.CTIME;
     OTHERWISE Set subfield indicator = Obstruction; <no more PWILST entries>
               Set clock bearing=0; <denotes null 18-bit subfield>
     SET entry SENT;

END subfield_18_bit_generation;
```

```
-------------------------------------------------------------------------------

PROCESS three_advisories_message_generation;

     REPEAT UNTIL (three fields generated);
          IF (first or second field)
               THEN PERFORM subfield_18_bit_generation;
               ELSE PERFORM subfield_12_bit_generation;
     ENDREPEAT:

     ADS=Three Advisories;
     IF (any subfield contains Threat, Terrain, Airspace, or Obstacle types)
          THEN SET Priority bit in message;
          ELSE CLEAR Priority bit in message;
     Move to uplink buffer;

END three_advisories_message_generation;
```

```
--------------------------------------------------------------------------------
PROCESS three_advisories_message_generation;

    REPEAT UNTIL (three fields generated);
        IF (first or second field)
            THEN PERFORM subfield_18_bit_generation;
            ELSE PERFORM subfield_12_bit_generation;
    ENDREPEAT;

    ADS=Three Advisories;
    IF (any subfield contains Threat, Terrain, Airspace, or Obstacle types)
        THEN SET Priority bit in message;
        ELSE CLEAR Priority bit in message;
    Move to uplink buffer;

END three_advisories_message_generation;
```

```
---------------------------------------------------------------------------------
FUNCTION FTAT    <first_time_advisory_transmitted>
   IN (PWILST entry)
   OUT (first-time indication);

     IF (Type has not changed)
          THEN Not first-time;
          ELSE First-time;    <new entry OR change of type>

END FTAT;
```

```
------------------------------------------------------------------------------------------
FUNCTION FTAT    <first_time_advisory_transmitted>
   IN (PWILST entry)
   OUT (FTAT):<BIT>

      IF (entry TYPE=entry OLD_TYPE)
           THEN CLEAR FTAT;
           ELSE SET FTAT;

 END FTAT;
```

17. FAILURE MODE OPERATION

17.1 Failure Provisions in ATARS Design

Protection against numerous types of failures is incorporated in the ATARS system design. Specific features are provided to cope with the following ATARS-specific failures:

1. Failure to receive a target report from the local sensor

2. False altitude or track association in target report from local sensor

3. Failure to deliver traffic or resolution advisory by local sensor

4. ATARS selects a resolution advisory which is incompatible with an aircraft's existing resolution advisories

5. Failure of a ground communication channel between sensors

6. Where a ground communication channel is not provided, or has failed, ATARS selects resolution advisories, not knowing the pair of aircraft is already being resolved by another ATARS site

7. Failure of the DABS sensor at a single site

8. Failure of the ATARS function at a single site

9. Catastrophic failure of an ATC facility

Logic for items 7 and 8 is contained in this section. The features which accommodate the other items are found in other sections of this document. All the capabilities are discussed here to summarize the robust nature of the overall design.

17.1.1 Missing Target Report

If the local sensor misses a target report on an aircraft, it requests a report from an adjacent sensor. ATARS performs coordinate and time conversion for the remote report and uses it to update the track for the aircraft. If the aircraft is ATARS equipped, ATARS requests a RAR report from the remote sensor. Even if that sensor was not previously reading the aircraft's RAR, the remote sensor may do so.

When a sensor detects an aircraft passing into its antenna's zenith cone ("cone of silence"), it requests an adjacent sensor to provide target reports continuously until told to stop. In a like manner, ATARS requests RAR reports from the adjacent sensor, for an equipped aircraft, to be provided until told to stop. The adjacent sensor "borrows" the original site's ATARS ID during this condition.

If no target report is obtained from any sensor (e.g., no adjacent sensor covered the aircraft's location or no ground communication channel is operating), ATARS coasts the track using its last estimates of position and velocity. Full conflict detection continues. If no report is received by a predetermined time, ATARS drops the track.

If a position report is received with altitude missing, the altitude estimate is coasted. If a target report is received but RAR data is missing, ATARS assumes the last known RAR contents are unchanged, rather than assuming an empty RAR.

17.1.2 Target Report Contains False Altitude or Track Association

ATARS maintains tracks on aircraft in its service area which are independent of DABS surveillance tracks. Since the requirements of a collision avoidance system differ from a surveillance system, ATARS uses its own criteria for establishing or dropping tracks.

ATARS performs a reasonability check on each altitude report. If unreasonable, the altitude report is ignored. If a falsely decoded altitude is sufficiently reasonable to be accepted, it is smoothed by the tracker and thus is unlikely to seriously affect ATARS service.

17.1.3 Sensor Fails to Deliver Traffic or Resolution Advisory

Although the DABS data link is very reliable, the sensor may occasionally fail to deliver traffic or resolution advisories. When a target report was received, but part or all of the ATARS messages were not delivered during the beam dwell, ATARS has good reason to believe that the aircraft is still visible to the sensor. ATARS simply computes updated advisories for the next scan. In the meantime, the avionics retains the existing advisories until they are updated, or until a time out several scans in length has expired. When an ATARS uplink message was attempted and not even a target report was successfully achieved by the local sensor, ATARS immediately attempts to deliver its messages through one (and only one) adjacent connected sensor. The adjacent sensor borrows the local ATARS ID, regardless of whether or not its own ATARS is providing service to the aircraft. The DABS multi-link protocol may not be used for ATARS uplink messages.

17.1.4 ATARS Selects Incompatible Resolution Advisory

ATARS resolution logic always selects new resolution advisories compatible with an aircraft's existing resolution advisories. However, if an existing advisory is not known to the ground, an imcompatible (i.e., contradictory) sense advisory could be uplinked. This could happen if a BCAS outside ATARS coverage (called a "pop-up" threat) initiated resolution with the subject aircraft since the last RAR downlink; or if another ATARS site, unconnected by a ground communications link, resolved another conflict since the last RAR downlink.

Any incompatible advisory is rejected by the ATARS avionics. The RAR performs a compatibility check for every uplinked advisory. If multiple advisories are beng uplinked, all compatible advisories will be accepted even if others are incompatible and rejected. ATARS reads down a copy of the RAR contents as they existed at the time the RAR was accessed. ATARS duplicates the avionics' compatibility logic to immediately determine whether each of its uplink advisories will be accepted. For any which are found incompatible, new advisories are recomputed for delivery the next scan, taking account of the updated copy of the RAR contents. This logic is described in Section 5.2.

17.1.5 Failure of Ground Communications Channel

A ground communications channel between sensors is not a critical element for ATARS operation. Where a network of more than two DABS sensors exists, the loss of a ground channel prompts DABS network management to establish alternate message paths. Whenever this succeeds, the ground channel failure is transparent to ATARS.

When ATARS becomes unconnected to a neighbor, some degradation of service will occur, since remote reports, cone of silence coverage, and remote uplink become unavailable. However, the majority of service, including multi-site coordination of conflict resolution, continues unaffected. The RAR is used as the coordinating device for all resolution, and responsibility is unchanged.

17.1.6 Resolution Duplicating That Provided by Another Site

Whenever a ground communication channel is available, positive coordination between sites assumes that only one site at a time issues resolution advisories to a particular pair of aircraft. However, when no channel is provided, or when the channel has failed, such duplicate service may be attempted in certain situations. This primarily happens when a DABS-ATCRBS pair moves into the low altitude part of an ATARS seam; or for unusual situations such as when a DABS-DABS pair flies into an ATARS seam and site responsibility changes just before the conflict begins, and one site has been unable to read down the ATARS site ID bits for at least one scan since they changed. The latter situation is a compound event of low probability.

The principal case is handled by having the site designated "lower priority" evaluate the other site's advisory to the DABS aircraft. If this appears adequate, the site yields responsibility to the higher priority site. In any event, the compatibility logic within the RAR prevents contradictory advisories from being posted.

17.1.7 Failure of the DABS Sensor

The DABS sensor is complex and may fail in a variety of ways, many of which are beyond the scope of this document. Any failure which causes the local ATARS function to fail is treated in the next Section.

If only the surveillance function or RF channel fails, so that ATARS continues to operate, data from remote sensors may be used as described in Section 17.1.1. In this case, ATARS attempts to provide service in its usual area, but is limited to servicing those aircraft seen by adjacent connected sensors. It is the responsibility of the local DABS to request the remote surveillance.

17.1.8 Failure of ATARS Function

Any network of neighboring DABS sensors may take advantage of overlapping DABS coverage for the purpose of allocating replacement coverage for a failed ATARS site. Section 17.3 discusses the means for recognition of a failed ATARS and the action to be taken.

17.1.9 Failure of the ATC Facility

ATARS normally serves controlled aircraft only as a backup to ATC. When aircraft come into conflict in sufficiently hazardous situations, ATARS issues traffic and resolution advisories. This action is performed routinely, and in the event of a catastrophic ATC failure, ATARS continues to provide full service.

17.2 Function Status Processing

Each DABS sensor contains a performance monitoring function. Once per scan, the sensor sends sensor status and ATARS Status Messages to all adjacent sensors. ATARS is only interested in the status ("operational" or "failed") of the ATARS function of each of its neighbors. Whenever such Status Messages are received, the Non-surveillance Message Processing Task (Section 5.1) calls the Remote Function Status Routine. This routine examines all remote ATARS Status Messages which have been queued since the last execution of the routine. The routine maintains a function failure table indicating the status of each remote ATARS. However, the logic only accommodates a single remote ATARS failure.

17.3 ATARS Backup Mode

Upon recognizing the first such failure, the Backup Mode Initiation Process is performed. This process replaces the ATARS service map with the backup service map corresponding to the failed ATARS. Tables are stored to indicate the appropriate service map to use after the failure of any neighboring site. The table MAPTBL (i) contains the list of map vertices corresponding to site (i) failing. The value i = 0 is used for the normal mode of operation. Similarily, the table MASTRTBL(i) contains bits to indicate whether or not own-site becomes the master site upon the failure of site (i). If Own-site becomes Master for more than one neighbor's failure (but always one at a time), another table CTRTBL(i) stores the map vertices for the center zone areas. ATARS does not assume a remote ATARS failure when a communication line fails. A separate register keeps track of currently connected sites. This is used in various places in the logic.

When an "operational status" indication is received in a message from a previously failed ATARS, the Backup Mode Termination Process is executed. This routine merely reinstates the normal service area, and resets the Backup and Master flags. It is not necessary to send Conflict Tables to the recovered site, since that site should be able to immediately read down aircraft RAR's, and may request Conflict Tables from its neighbors over ground lines. When the (smaller) normal service map is reinstated, aircraft outside this map will be dropped from ATARS service in the normal way, just as if they had flown out of the service zone.

17.3.1 Backup ATARS Service Areas

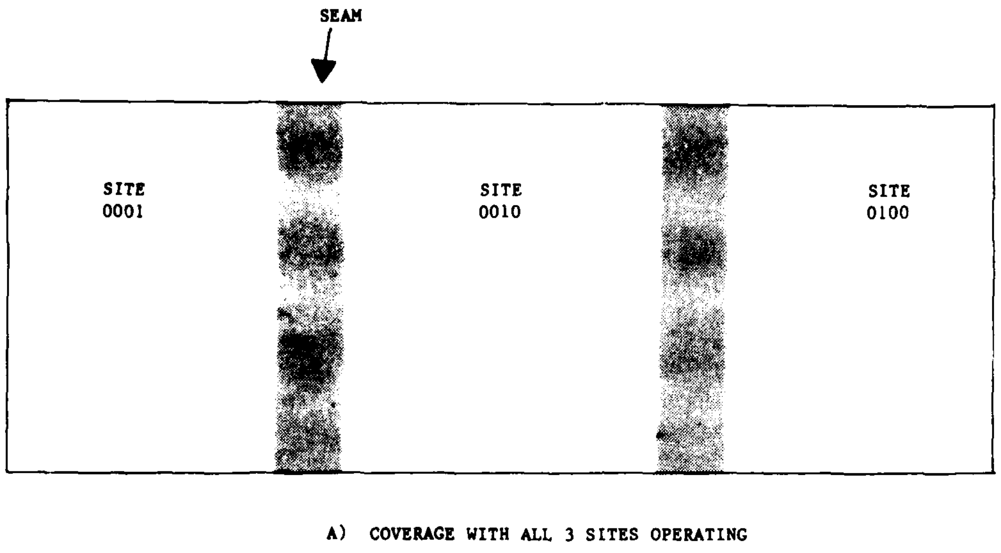
In a region having several DABS sensors located within reasonable proximity, some flexibility may exist in drawing the ATARS map boundaries. The choice of boundaries will take into account geographical coverage, terrain features and the expected traffic load for the sensors. Upon the failure of one ATARS site, other sensors with overlapping coverage may be capable of replacing the failed ATARS. If the failed site was serving a large load of traffic, no single neighbor may have sufficient capacity to absorb it all. Therefore, a better strategy, where geographic coverage and terrain features permit, is to divide the failed site's service area among several neighbors.

An example of this operation is depicted in Figure 17-1. Here, when the site with ID 0010 fails, its two neighbors each expand their coverage and share the failed site's area. The two surviving sites' maps should provide an overlap (seam) of the usual width. In this case, no "master" site or "center" zone (see below) is required. Both surviving sites operate normally, and treat newly acquired aircraft in their expanded service areas in the same manner as any aircraft which has just entered the normal service area. Any of these aircraft having RAR entries created by the failed site will soon have them released by the avionics time out.

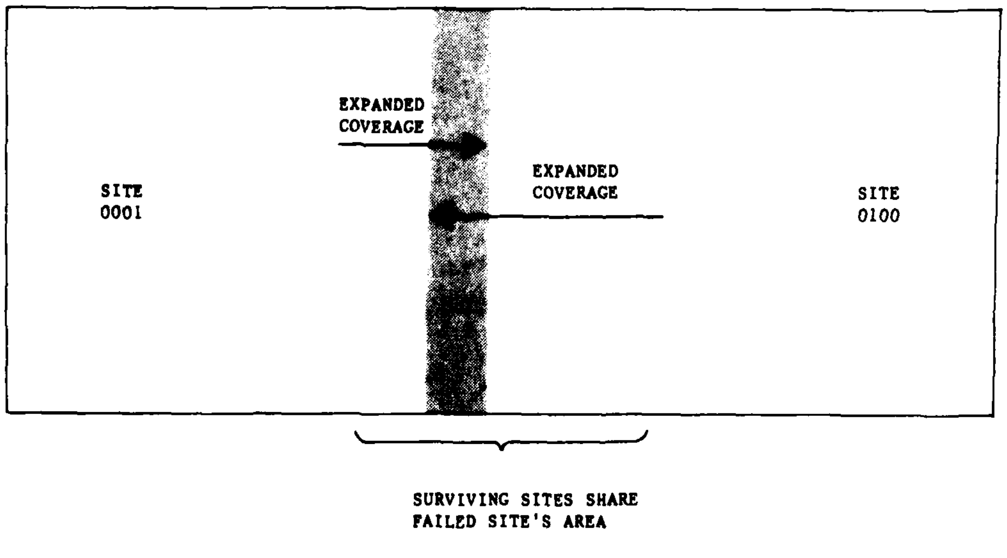
The surviving sites may not be connected with ground communications lines. In this case, all coordination is performed through the transponders using the RAR features, as explained in Section 14.2.

17.3.2 Master Site

In certain configurations of ATARS sites, the simple procedure described above cannot be used. Since only four distinct ATARS ID's are assigned, the failure of an ATARS may cause two sites assigned the same ID to become adjacent, if the most desirable backup service map were implemented. Since the multi-site protocol does not permit this condition, several alternatives are available. Using another neighbor to cover the failed area may be feasible, but geographic considerations may prohibit this choice. Leaving a coverage gap without ATARS is very undesirable. The solution implemented in this design is to designate one of the surviving sites as the "Master" site. This should be the site with the best geographical coverage of the part of the failed area that separates the two sites having the same ATARS ID. The master site then continues to serve its own



A) COVERAGE WITH ALL 3 SITES OPERATING



B) COVERAGE WHEN SITE 0010 FAILS

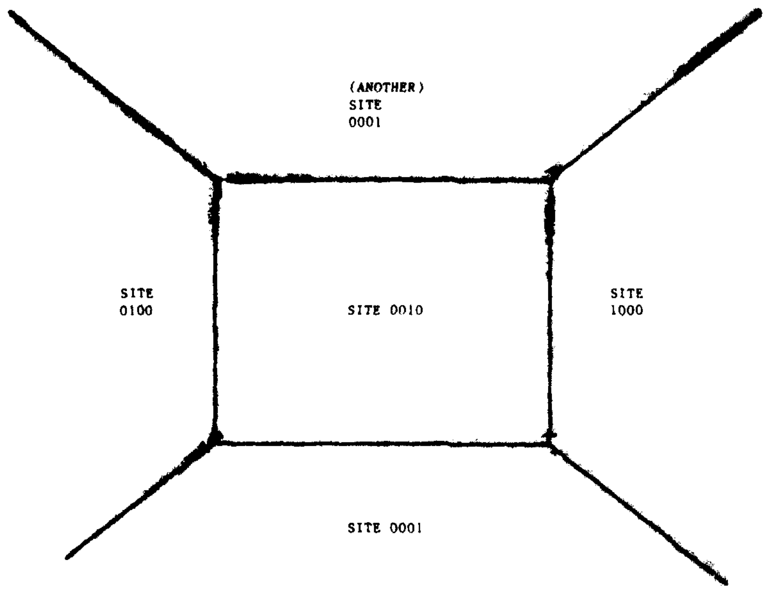
FIGURE 17-1
EXAMPLE OF AREA WHERE CENTER ZONE NOT USED FOR BACKUP COVERAGE

ATARS area using its own ID, and also serves a small "center" zone within the failed site's area, borrowing the failed site's ATARS ID to send to aircraft in this area. This is illustrated in Figure 17-2. The center zone should be made small, so all sites may use their own ID in as large an area as possible, but sites with like ATARS ID's must be separated by more than the usual seam width.

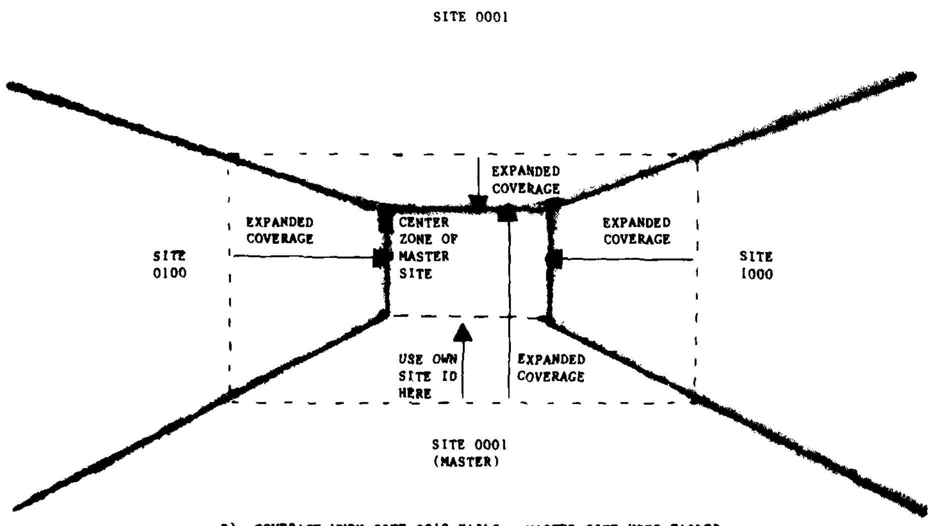
The master site performs an extra masking (in this backup mode) to decide which of the aircraft in its expanded coverage are in the center zone and thus are to receive the failed site's ID. The master site still uses its own sensor to service both of its areas, sending different ATARS ID's to aircraft according to their location. The center zone is mapped to have the usual overlap (seam) with all of its neighboring sites except the master site. No overlap is provided between Master's own area and its center zone. The master ATARS keeps aircraft in both of its areas in the same data base, and is able to treat the boundary between its two areas as "soft". This means that if an aircraft receiving resolution crosses this boundary, the master site may change the site ID sent to that aircraft, while continuing to send the other ID to the other aircraft until it too crosses the boundary.

17.4 Pseudocode for Failure Mode Operation

The pseudocode follows. The Remote Function Status Routine is called from the Non-Surveillance Message Processing Task. Upon initiating the Backup mode, the ID of the failed site is used as an index to the tables of service areas, Master status flag settings, and Center zones. An index of zero is used for the Normal mode of operation.



A) COVERAGE WITH ALL 5 SITES OPERATING



B) COVERAGE WHEN SITE 0010 FAILS. MASTER SITE USES FAILED SITE'S ID IN CENTER ZONE

FIGURE 17-2
EXAMPLE OF AREA WHERE CENTER ZONE IS USED FOR BACKUP COVERAGE

PSEUDOCODE TABLE OF CONTENTS

```
------------------------------------------------------------------------------------
ROUTINE REMOTE_FUNCTION_STATUS
   IN (status message from remote ATARS, local DABS)
   INOUT (Master flag, Backup flag, service map, state vectors, conflict tables);

     IF (msg says remote ATARS is operational)
          THEN IF (this remote ATARS was failed)
                    THEN PERFORM backup_mode_termination;
     ELSEIF (msg says remote ATARS has failed)
          THEN IF (own ATARS not in Backup mode)
                    THEN PERFORM backup_mode_initiation;
     <note this logic only handles one site failure at a time>
     ELSEIF (msg says remote site link failed/recovered)
          THEN Update connected site register;
     ELSEIF (msg says own DABS not operational)
          THEN Halt own ATARS processing;
               LOOP:
               EXITIF (startup msg received from DABS);
               ENDLOOP;
     OTHERWISE:  <own DABS OK>

END REMOTE_FUNCTION_STATUS;
```

```
------------------------------------------------------------------------------------

ROUTINE REMOTE_FUNCTION_STATUS
   IN (status message from remote ATARS, local DABS)
   INOUT (SYSVAR, state vectors, conflict tables);

     IF (msg says remote ATARS is operational)
          THEN IF (SYSVAR.FAILED EQ this remote site)
                    THEN PERFORM backup_mode_termination;
     ELSEIF (msg says remote ATARS has failed)
          THEN IF (SYSVAR.BACKUP not set)
                    THEN SYSVAR.FAILED=failed site_ID;
                         PERFORM backup_mode_initiation;
     <note this logic only handles one site failure at a time>
     ELSEIF (msg says remote site link failed/recovered)
          THEN Set connected site bit to status indicated in msg;
     ELSEIF (msg says own DABS not operational)
          THEN Halt own ATARS processing;
               LOOP;
               EXITIF (startup msg received from DABS);
               ENDLOOP;
     OTHERWISE;  <own DABS OK>

END REMOTE_FUNCTION_STATUS;
```

```
------------------------------------------------------------------------------------------
PROCESS backup_mode_termination

     Activate normal ATARS service map;
     Deactivate Center zone map;
     CLEAR Master flag;
     CLEAR Backup flag;

END backup_mode_termination;
```

```
-----------------------------------------------------------------------------------

PROCESS backup_mode_termination

     SYSVAR.FAILED=0;
     SYSVAR.MAPPTR=SYSTEM.MAPTBL(0);   <reactivate normal ATARS service map>
     SYSVAR.CTRPTR=$NULL;
     CLEAR SYSVAR.MASTER;
     CLEAR SYSVAR.BACKUP;

END backup_mode_termination;
```

```
--------------------------------------------------------------------------------

PROCESS backup_mode_initiation

     Set Backup flag, Master flag if required for this site failure;
     IF (Master flag set)
          THEN Activate Center zone map for this failure;
     Activate backup ATARS service map for this failure;
     REPEAT WHILE (more pair records showing failed site in charge):
          Select next pair record;
          IF (state vectors exist for both aircraft)
               THEN CLEAR failed site's bit in GEOG of both aircraft;
                    Set handoff bit in pair record;
               ELSE CALL PAIR_RECORD_DELETION;
     ENDREPEAT;

END backup_mode_initiation;
```

```
-------------------------------------------------------------------------------------
PROCESS backup_mode_initiation

     SET SYSVAR.BACKUP;
     SYSVAR.MASTER=SYSTEM.MASTRTBL(SYSVAR.FAILED);
     IF (SYSVAR.MASTER EQ STRUE)
          THEN SYSVAR.CTRPTR=SYSTEM.CTRTBL(SYSVAR.FAILED);
                    <activate Center zone map for this failure>
     SYSVAR.MAPPTR=SYSTEM.MAPTBL(SYSVAR.FAILED);
                   <activate service map for this failure>
     REPEAT WHILE (more pair records with ATSID EQ failed site);
          Select next PREC;
          IF (state vectors exist for both PREC.PAC1 and PREC.PAC2)
               THEN CLEAR failed site's bit in SVECT.GEOG of both aircraft;
                    SET PREC.HDOFF;
               ELSE CALL PAIR_RECORD_DELETION
                       IN (PREC)
                       INOUT (confl. table, state vectors);
     ENDREPEAT;

END backup_mode_initiation;
```

18. PERFORMANCE MONITOR

This section develops functional requirements for the Status Monitoring and Reporting (SMR) Function of ATARS. The ATARS SMR provides start and stop control indication to the local ATARS function executive processing routine. In addition the SMR monitors the status of the various functional modules, buffers and files of ATARS. The SMR constructs and transmits messages indicating the status of ATARS.

These messages are routed to the ATARS Non-surveillance Buffer. The DABS Performance Monitor reads these messages to determine the local ATARS status. The DABS Performance Monitor also transmits these messages to NAS facilities (ATC, RMM) and clears the buffer once a scan.

The Status Messages sent to DABS have limited capacity for failure reporting. If the reportable status conditions exceed this capacity then an indication of this condition is sent in the message. In accordance with this the SMR is also capable of transmitting a complete list of status conditions to a requesting NAS facility via the ATC Coordination Buffer.

The following paragraphs describe the inputs, processing function, and outputs of the ATARS SMR.

18.1 Status Monitoring and Reporting Function Inputs

DABS Status

The local DABS sensor status as determined by the DABS Performance Monitor is reported (Status Message) to the SMR via the Non-surveillance Buffer. This status will allow the SMR to control ATARS operation depending on the operational status of the local DABS sensor. This message may also report adjacent ATARS or sensor status. Logic for such reports is contained in Section 17.

ATARS Functional Failure Conditions

These inputs consist of various failure indicators from the functional modules of ATARS. These failure indicators indicate memory utilization failure, processing truncation, and software failures.

Memory Utilization Failure Indication

There are three catagories of failure indicators for memory utilization. These are: output buffer overflow, file more than 90% full, and no more space in file. These are described below.

Output Buffer Overflow

This condition is reported for all ATARS buffers (see Figure 3-3) and specifies the applicable buffer.

File More than 90% Full

The SMR monitors the degree of ATARS file utilization and reports any condition of over 90 percent full for the following files:

1. Central Track Store
2. Conflict Tables
3. PWILST's
4. Encounter Lists

No More Space in File

This condition is reported for the same files listed in paragraph above.

Processing Truncation

The ATARS executive program reports any occurrence of an ATARS task being truncated due to a processing timing deadline. The report identifies the task and the sector of data for which processing was truncated.

Software Failures

This condition is unusual and is detected with the use of sector flags or other suitable indicator. The indicator shall be set by each of the ATARS functions when reporting their status for the SMR. When the SMR reads the status each scan it updates this indicator. If the SMR finds any indicator not properly updated, a failure is declared.

Request Full ATARS Status Control Message

This message from NAS to ATARS controls ATARS SMR reporting. The format is contained in Reference 8.

This message is processed by the SMR as described in Section 18.2. The following sections describe the control field in this message.

Field CTLTX

This two bit field is for transmission control of all active condition codes to a requesting NAS facility. The interpretation is as follows:

00 = transmit all condition codes for one scan
01 = transmit all condition codes repeatedly for NSMR scans (a system parameter) unless told to stop
10 = stop transmission of all condition codes
11 = not used

18.2 Status Monitoring and Reporting Function Processing

This section describes the processing tasks to be performed by the SMR.

18.2.1 Local ATARS Processing Control

The SMR controls the commencement and cessation of local ATARS processing under the following conditions.

1. DABS/ATARS local sensor start up: (Reference 1)

   a. Cold start
   b. Warm start

2. Failure of the DABS portion of the local sensor

3. Failure of the ATARS portion of the local sensor

Commencement of ATARS activity will occur under item 1 above when all the ATARS functions have begun to set their software failure check indicators.

Cessation of ATARS activity will occur for items 2 and 3 above. An ATARS failure will be declared after one scan of operation with a red condition indicator if the red condition is not cleared.

18.2.2 Message Generation and Processing

The ATARS SMR processes the indicators and control message decribed in Section 18.1. Two outputs are constructed in each scan (see Reference 1):

1. ATARS Status Message to sensor. This message is used to notify the sensor of the operation or failure of the local ATARS function. The sensor is expected to utilize this data in its Status Message sent to adjacent ATARS functions.

2. ATARS Status Message to ATC. This output is one of the three types of messages listed below. Their formats are defined in Reference 8, with additional detail in Section 18.3 below. These messages convey to ATC a more detailed description of the local ATARS status. The messages are:

   ATARS Green Condition
   ATARS Yellow Condition Codes
   ATARS Red Condition Codes

   These messages may be read by a local Status Monitor Display for the sensor.

The ATARS SMR maintains a current determination of the local ATARS status. This status may be classified as "Green" (normal operation), "Yellow" (marginal operation), or "Red" (Failed). The ATARS status flag shall be reported as "normal operation" when the local status is Green or Yellow, and as "failed" when the local status is Red (see Section 18.3.3).

The categories of indicators and assigned number of scans of consecutive occurrences to produce a Yellow or Red status condition are listed below:

	CATEGORY	NUMBER OF SCANS TO DECLARE STATUS	
		Yellow	Red
1.	Output Buffer Overflow (each buffer)	2	-
2.	File 90% Full (each file)	2	-
3.	No Space on File (each file)	1	2
4.	Processing Truncation (each task)	1	-
5.	Software Failures (each task)	1	2

Also, if two or more files are full in one scan, Red status is declared.

The SMR shall construct a list of active Yellow and Red condition codes. These codes identify all buffers, files, tasks, and services which meet an identified failure condition, and specify the failure condition for each.

If none of the above categories of indicators have occurred, then the local status is Green. If any Red condition has occurred, the local ATARS status is Red. If no Red condition has occurred, but any Yellow condition has occurred, the local ATARS status is Yellow.

The ATARS Status Message to ATC shall be determined from Table 18-1. The following paragraphs describe these messages and specify the processing to be performed when the number of active condition codes exceeds the number of available fields in the indicated mesage.

ATARS Green Condition Message

The Green condition implies ATARS is fully operational.

ATARS Yellow Condition Codes Message

This Yellow condition is a warning that ATARS is functioning at a reduced service level. The Yellow Condition Codes Message, with field ALL = 0, may specify up to 25 Yellow condition codes. These condition codes describe specifically how ATARS is operating at a reduced level.

If there are more than 25 Yellow condition codes active, the Last Condition Code Message/Overflow (LM/OVF) flag is set in the message. The monitoring facilities (ATC, RMM) may request full status reporting using the message specified in Section 18.1. The SMR performs the requested action in the message.

ATARS Red Condition Codes Message

The Red condition indicates some portion of the ATARS software has failed. After the detection of this code the SMR halts local ATARS processing according to the rules in Section 18.2.1. However, the SMR continues to transmit upon request any existing Yellow condition codes as well as the Red condition codes which were active as of the time of shut down.

TABLE 18-1

SELECTION OF STATUS MESSAGE TO ATC

LOCAL ATARS STATUS	ALL CODES REQUEST STATUS	MESSAGE SENT
Green	Any	ATARS Green Condition
Yellow	Yes	ATARS Yellow Condition Codes (field ALL = 1)
Yellow	No	ATARS Yellow Condition Codes (field ALL = 0)
Red	Yes	ATARS Red Condition Codes (field ALL = 1)
Red	No	ATARS Red Condition Codes (field ALL = 0)

If there are more than five Red condition codes active, or any Yellow condition codes in addition to the Red condition codes, the LM/OVF flag is set in the Red Condition Codes Message. ATC must request all condition codes in this case to be sent any Yellow codes. Yellow codes would be sent in a separate message, even if the Red Condition Codes Message had unused fields.

18.2.3 Request Full ATARS Status Control Message Processing

The actions to be taken by the SMR upon the receipt of this message depend on the specified transmission control (see Field CTLTX, Section 18.1) as follows:

Transmit All Condition Codes for One Scan

The SMR transmits to the requesting facility all the currently active condition codes (Red and Yellow) using as many messages as necessary. If there are none, the ATARS Green Condition Message is sent. The SMR uses the appropriate condition codes messages (field ALL = 1) for Red or Yellow codes. Each message specifies up to 50 condition codes. As many messages shall be transmitted as are needed to transmit all active condition codes. The Last Condition Code Message/Overflow bit is set in the last Red or Yellow code message.

Transmit All Condition Codes Continuously

Yellow and Red codes active during a scan are transmitted once a scan for as many scans as they exist, for a maximum of NSMR scans, with NSMR a site adaptable ATARS parameter. After NSMR scans, the "All Codes Request Status" for the requesting facility is reset until a new request is received. If there are no Yellow or Red codes active at the time of receipt, the ATARS Green Condition Message is sent only once and the "All Codes Request Status" is immediately reset. If, prior to NSMR scans, all Yellow or Red condition codes revert to Green, the ATARS Green Condition Message is sent once and the "All Codes Request Status" is immediately reset for each requesting facility.

Stop Continuous Transmission of All Codes

When the SMR has been transmitting all condition codes continuously, this message resets the "All Codes Request Status" for the requesting facility. Otherwise the SMR ignores the message.

18.3 Status Monitoring and Reporting Function Outputs

The outputs of SMR processing consist of the control messages to allow commencement or cessation of local ATARS processing, and the messages listed in Section 18.2.2. The following paragraphs supplement Reference 8, to define the contents of the message fields.

18.3.1 Fields in Output Messages

Field ALL

When this bit, All Condition Codes Tranmission, is set to 1, the message is part of a group (one or more) of messages containing all active ATARS Yellow or Red condition codes. When set to 0, only one ATARS Yellow or Red Condition Codes Message is being transmitted. The existence of any untransmitted condition codes is indicated by the Last Condition Code Message/Overflow (LM/OVF) field defined below.

Field LM/OVF

This bit, Last Condition Code Message/Overflow, has a dual purpose depending on the value of the field ALL, described above.

When ALL = 0, this bit is interpreted as the overflow indicator bit for ATARS Yellow and Red Condition Codes Messages. LM/OVF set to 1 indicates more condition codes are currently active than are being transmitted in a single message. The maximum number of Yellow and Red condition codes that can be transmitted under this setting of ALL = 0 is 25 and 5 respectively.

When ALL = 1, this bit is interpreted as the last condition code message indicator flag and is set to 1 in the last (of one or more) condition codes message. This, when combined with the field ALL, allows transmission of all active condition codes when more than one message is required.

Field #AY

This field, used in the Yellow Condition Codes Message, is a binary integer indicating the number of Yellow condition codes contained in the message. This number is a maximum of 25 when field ALL = 0, and a maximum of 50 when field ALL = 1.

Field AY(1)...AY(n)

These fields contain the Yellow condition codes as defined in Section 18.3.2. These fields are sorted from lowest to highest binary value, the lower value codes being transmitted first.

Field #AR

This field, used in the Red Condition Codes Message, is a binary integer indicating the number of Red condition codes contained in the message. This number is a maximum of 5 when field ALL = 0 and a maximum of 50 when field ALL = 1.

Fields AR(1)...AR(n)

These fields contain the Red condition codes and are defined in Section 18.3.2. These fields shall be sorted from lowest to highest binary value, the lower value codes transmitted first.

18.3.2 Formats of Condition Codes

The general format of the condition codes is as follows:

CATEGORY	CONTENT
6	16

The assigned categories and formats are described in the following paragraphs. Each condition code may be used as a Yellow or Red code in the message fields listed in Section 18.3.1. The placement in the proper message corresponds to the status determination described in Section 18.2.

The formats of the condition codes for each assigned category are as follows:

OUTPUT BUFFER OVERFLOW

000100	SP	BUFFER
6	8	16

FILE MORE THAN 90% FULL

000110	SP	FILE
6	8	16

NO MORE SPACE IN FILE

```
000101    SP   FILE
_________________________
     6     8          16
```

PROCESSING TRUNCATION

```
000010    SECTOR    FUNCTION
______________________________
     6        10          16
```

SOFTWARE FAILURES

```
000001              SP             FUNCTION
_____________________________________________
     6              10                    16
```

Field BUFFER

This field identifies a buffer of concern as follows:

Coding		Buffer
0000	0001	ATARS-ATARS Coordination Buffer
0000	0010	Uplink Message Buffer
0000	0011	Non-surveillance Buffer
0000	0100	ATC Coordination Buffer
0000	0101	RAR Buffer
0000	0110	Surveillance Buffer

Field FILE

This field identifies an ATARS file as follows:

Coding		File
0000	0001	Central Track Store
0000	0010	Conflict Tables
0000	0011	PWILST's
0000	0100	Encounter Lists
0000	0101	ATARS Sector Lists
0000	0110	X/EX Lists
0000	0111	Potential Pair List
0000	1000	Resolution Pair Acknowledgement List
0000	1001	Controller Alert List
0000	1010	Deletion List

Field SECTOR

This 4-bit field ide: :ifies an ATARS sector of data which was not fully processed.

Note: Each sector of data that is not fully processed shall be separately identified by an active condition code.

Field FUNCTION

This field identifies a function of ATARS in a condition code. The values and associated functions are as follows:

VALUE		FUNCTION
000	001	Master Resolution (Normal) Task
000	010	Master Resolution (Delayed) Task
000	011	Data Link Message Construction Task
000	100	Coarse Screen Task
000	101	Traffic Advisory Task
000	110	RAR Processing Task
000	111	Aircraft Update Processing Task
001	000	Track Processing Task
001	001	Seam Pair Task
001	010	Detect Task
001	011	Conflict Pair Cleanup Task
001	100	State Vector Deletion Task
001	101	Request and Process Remote Conflict Tables Task
001	110	Conflict Resolution Data Task
001	111	Resolution Notification Task
010	000	Incoming Seam Pair Request Processing and Reply Task
010	001	Surveillance Report Processing Task
010	010	Non-surveillance Message Processing Task
010	011	New Aircraft Processing Task
010	100	Terrain/Airspace/Obstacle Avoidance Task
010	101	Resolution Deletion Task

18.3.3 ATARS Normal Operation/Failure Flag

This flag in the ATARS Status Message to the sensor is used by the DABS Performance Monitor in the DABS Status Message described in Reference 8. The SMR transmits this bit according to the following rules:

Bit = 1	This setting is used when the SMR has declared local ATARS status to be Green or Yellow.
Bit = 0	This setting is used when the SMR has declared local ATARS status to be Red (Failed).

19. DATA EXTRACTION FUNCTION

The purpose of the Data Extraction Function of ATARS is threefold: to provide the capability to conduct detailed analysis of conflict scenarios, to locate and correct erroneous code (debug), and to serve as an operations log recording important events within ATARS. A well designed data extraction capability for conflict scenario analysis should be able to incorporate the needs of the remaining two functions.

19.1 Information Recorded

The information required for analysis is (1) the results of important calculations, such as TH and TV, and (2) the logic paths taken within ATARS. Table 19-1 gives the parameters to be saved on a scan-by-scan basis for off-line analysis. Table 19-2 lists the logic paths taken in the code to be recorded. The path checkpoints are an attempt to give in summary the reason for a particular ATARS action, without the detailed decision process explicitly documented. For example, if an immediate resolution advisory is requested by the Detect Task, one wants to know which of the seven ways this can happen. A possible implementation is to have a variable with eight values, indicating how the immediate RA was generated or set to zero indicating no action through these paths occurred.

Recording of all the information in Tables 19-1 and 19-2 at all times would be very cumbersome and perhaps affect the efficiency of the ATARS. So a selection control function is required to reduce the information extracted as described. Table 19-3 defines the allowable options for extracting data. Whenever data is to be saved, i.e., removed from the ATARS system, a comparison is made to see if it corresponds to that data to be recorded for the particular mode specified for the present ATARS configuration. More than one mode can be specified. Detect Task traffic advisory operations can be requested with Master Resolution Task operations, for example. When multiple modes are specified, it is of course not necessary to record the same information twice.

Judicious placement of the selection test in the ATARS code can limit the amount of testing done for extraction. It is not necessary to test for possible extraction every time a Detect Task flag is set. It is only necessary to test at the end of detection processing for a flag set condition, since the items being saved exist until this point. The actual time or location for extraction can occur in many places and is not specified. The only requirements of such extraction are that it be editable,

TABLE 19-1

INFORMATION INCLUDED IN DATA EXTRACTION

Aircraft Description Information

1. Aircraft ID - DABS ID or ATCRBS/Radar code with surveillance file number, any CREFX entries and the AC abbreviated field in the State Vector (ACAB)
2. Sector Time for Aircraft - system variable TEN
3. ACLASS - ATARS service class, as in State Vector
4. CUNC - aircraft control state, as in State Vector
5. ATSEQ - equipage type, as in State Vector
6. ACLP - aircraft climb performance, as in State Vector

Tracking Information

7. HMS - from State Vector
8. TURN - from State Vector
9. Tracked X, Y, Z - from State Vector
10. Reported range, azimuth and mode C altitude - from DABS/ATCRBS/Radar data block in State Vector
11. Velocity (XD, YD, ZD) of the aircraft - as in State Vector
12. LOFL - local data flag, as in State Vector
13. RMFL - remote data flag, as in State Vector

Detect Information

14. ENAT - from ELENTRY
15. DOT - from ELENTRY
16. MD2 - from ELENTRY
17. RANGE2 - from ELENTRY
18. TH - from ELENTRY
19. TV - from ELENTRY
20. RZ - as in Detect Task
21. VRZA - as in Detect Task
22. MULT - as in Detect Task
23. FAZ - from State Vector
24. AREA TYPES for both AC - from State Vector
25. TCONV - as in Detect Task
26. TCONH - as in Detect Task
27. TFPWIV - as in Detect Task
28. TFPWIH - as in Detect Task
29. TCMDV - as in Detect Task
30. TCMDH - as in Detect Task
31. TFIFRV - as in Detect Task

TABLE 19-1
(Continued)

32. TFIFRH - as in Detect Task
33. TIFRV - as in Detect Task
34. TIFRH - as in Detect Task
35. ICAFLG - from ELENTRY
36. CAFLG - from ELENTRY
37. FPWFLG - from ELENTRY
38. MTTFLG - from ELENTRY
39. FPIFLG - from ELENTRY
40. CMDFLG - from ELENTRY
41. FPWFLG - from ELENTRY
42. IFRFLG - from ELENTRY

Resolution Information

43. PSEP matrices (QSEP, before and after PSEP, HMD, VMDA, VMDB) - as in Master Resolution Task
44. Pair Record on entry to RAER - as in Master Resolution Task
45. SNGDIM - as in Master Resolution Task
46. Conflict Tables - as in Master Resolution Task
47. Feature Evaluation vs Resolution Advisory Set - as in Master Resolution Task
48. MANTM - as in Master Resolution Task (Structure MODVBL)

Domino Information

49. Coarse Screen Limits - as in domino logic
50. Potential Domino Conflict List - as in domino logic
51. Resolution Advisory Projected Position Table - as in domino logic

Multi-aircraft Logic Information

52. PSEP matrices - as specified in multi-aircraft logic (not the same as 43)
53. Resolution Advisory vs Feature Evaluation Table - as specified in multi-aircraft logic (not the same as 44)
54. Conflict Tables - as specified in multi-aircraft logic
55. Pair Records - as specified in multi-aircraft logic

Terrain Airspace Obstacle Avoidance Information

56. TALRT - as in Structure TAO
57. OALRT - as in Structure TAO
58. TCALRT - as in Structure TAO

TABLE 19-1
(Concluded)

Controller Alert Task Information

59. RALRT - as in Structure TAO
60. Conflict Resolution Data Message - as in Table 11-1
61. Resolution Notification Message - as in Table 11-2

TABLE 19-2

LOGIC PATH CHECKPOINTS

Detect Controller Alert Paths

a. ICAFLG is set (i.e., bypass 3/5 requirement) because:
   1. Immediate range and immediate altitude is satisfied for FAZ (in controller_alert_determination)[1]
   2. HPROX and VPROX satisfied (ibid)
   3. Dangerous maneuver detected (ibid)

   CAFLG is not set because:
   1. Prefiltering DOT test failed (in ac_converging_or_proximate_)
   2. Prefiltering horizontal test failed (ibid)
   3. Prefiltering vertical test failed (ibid)
   4. Controller alert inhibited, CAREQ not set (in controller_alert determination)
   5. Failed horizontal tests (ibid)
   6. Failed vertical tests (ibid)
   7. Failed miss distance test (ibid)

Detect Resolution Advisory Paths

b. MTTFLG is set (i.e., bypass 2/3 requirement) because:
   1. HPROX and VPROX set (in proximity_checks)
   2. TH below threshold and VPROX (ibid)
   3. HPROX set and TV below threshold (ibid)
   4. TH and TV below thresholds (ibid)
   5. Dangerous maneuver detected (in maneuvering_threat_logic)

c. CMDFLG is not set because:
   1. Prefiltering failed (in ac_converging_or_proximate)
   2. Failed horizontal tests for threat advisory (in THREAT_TAU_AND PROXIMITY_COMPARISONS
   3. Failed vertical tests for threat advisory (ibid)
   4. Failed miss distance tests for threat advisory in (ibid)
   5. Failed horizontal tests for resolution advisory (in RESOLUTION_ TAU_AND_PROXIMITY_COMPARISONS)
   6. Failed vertical tests for resolution advisory (ibid)
   7. Failed vertical divergence filter (ibid)

TABLE 19-2
(Concluded)

d. IFRFLG not set because:
   1. Prefiltering failed (in ac_converging_or_proximate)
   2. Failed horizontal tests for threat advisory (in THREAT_TAU_AND PROXIMITY_COMPARISONS)
   3. Failed vertical tests for threat advisory (ibid)
   4. Failed miss distance tests for threat advisory (ibid)
   5. Failed horizontal tests for resolution advisory (in RESOLUTION_ TAU_AND_PROXIMITY_COMPARISONS)
   6. Failed vertical tests for resolution advisory (ibid)
   7. Failed vertical divergence filter (ibid)

e. The resolution advisory origin was:
   1. Initial resolution selection caused call to RAER
   2. Initial resolution selection for IFR (VFR resolution previously selected) caused call to RAER
   3. Positive horizontal to negative transition caused call to RAER
   4. Negative horizontal to positive transition caused call to RAER
   5. Positive vertical to negative transition caused call to RAER
   6. Negative vertical to positive transition caused call to RAER
   7. Recomputation, MD2 less than PMD, caused call to RAER
   8. Recomputation, ALT less than PVMD, caused call to RAER
   9. Model validation logic caused call to RAER
   10. Negative vertical to VSL transition (RAER not called)
   11. Recalculation of incompatible resolution advisories caused call to RAER
   12. Conflict Resolution Data Task called RAER

[1]The capitalization conforms to the convention used in the pseudocode, where process names are all lower case and routine names are all upper case.

TABLE 19-3

SELECTION MODES FOR DATA EXTRACTION

MODE	DESCRIPTION	DATA RECORDED	INITIATION
1	Tracking Information	Table 19-1, #1-13	Whenever tracking is completed
2	Detect Summary	Table 19-1, #1-6, #35-42	Whenever a Detect Task flag is set
3	Detect CA Operations	Table 19-1, #1-36 Table 19-2, #a	Whenever CAFLG is set
4	Detect TA-prox Operations	Table 19-1, #1-42 Table 19-2, #b, c, d	Whenever PWIFLG is set
5	Detect TA-threat Operations	Table 19-1, #1-42 Table 19-2, #b, c, d	Whenever FPIFLG or FPWFLG is set
6	Detect RA Operations	Table 19-1, #1-42 Table 19-2, #b, c, d	Whenever CMDFLG is set
7	Master Resolution Summary	Table 19-1, #1-6, 46	Whenever resolution advisory is generated by RAER
8	Master Resolution Operations	Table 19-1, #1-13, #43-48 able 19-2, #e	Whenever resolution advisory is determined in Master Resolution Task
9	Domino Operations	Table 18-1, #1-13, #49-51	Whenever domino logic called
10	Multi-aircraft Operations	Table 19-1, #1-13, #52-55	Whenever multi-aircraft logic called

TABLE 19-3
(Concluded)

11	T/A/O Operations	Table 19-1, #1-13, #56-59	Whenever TALRT, OALRT, TCALRT, RALRT is set
12	Controller Alert Operations	Table 19-1, #1-13, #60-61	Whenever message sent
		Table 19-2, #e	Whenever RAER is called from Conflict Resolution Data Task

for efficiency and ease of use, and contain all information necessary for generation of the Data Analysis Summary Chart or any subset of the chart. This form is discussed in Section 19.3.

19.2 Scope of Design and Application

The data extraction system presented has minimal impact on the ATARS software. The data is "dumped in real time". All information required to decide if the data need be recorded is available at the time for recording without modification to the present system. An apparent limitation of the design presented, is the inability to record at all times why a particular flag was not set in the Detect Task. In order to do this, the implications would be:

1. Record every time a flag is not set.

   Disadvantage - Massive data handling and storage problems

   Advantages - Guaranteed knowledge of reason for condition desisting
   - Ability to generate all parts of data analysis summary chart

2. Record whenever flag is not set after being set on previous scans.

   Disadvantage - Information must be made available from scan to scan in the Detect Task
   - Global memory storage required on a per pair basis

   Advantages - Guaranteed knowledge of reason for condition desisting
   - Ability to generate all parts of data analysis summary chart
   - Introduction of global scan-to-scan storage for decisions would allow much more sophisticated data extraction designs

While it is not possible to guarantee recording of the reason for "not acting" under the present design, it is possible to arrange the edit mode options to cover most possibilities. For example the data analysis summary chart requires the path information in Table 19-2 items c and d, i.e, why is ATARS not giving a resolution advisory? Requesting mode 4 will record path information whenever the PWIFLG is set. Under normal conditions a resolution advisory will be requested after the PWIFLG is set and end before the PWIFLG goes off. Therefore, the resolution advisory path information is available for recording and analysis with mode 4. It is for this reason that modes 4, 5, and 6 record the same information but are initiated by different events.

Table 19-4 gives the intended use of the mode option in recording information.

19.3 The Data Analysis Summary Chart

An example of the data analysis chart is depicted in Figure 19-1. Values are provided only to illustrate the use and meaning of the various data items. This format and content have proven to be most satisfactory in studying aircraft encounters. Any encounter summaries generated from the ATARS using the specified data extraction techniques will be in this format. Notice that the data is only for one aircraft pair for the duration of an encounter. The extraction task is recording information for many encounter pairs on a scan-by-scan basis. A formatting program must first sort the recorded information by aircraft ID's. The scan-by-scan sequence can be attained by maintaining the exact sequence of the dump for aircraft and labelling each new recording of the same data item as a new scan. Alternately, the time recorded with each extracted data unit, can serve as a scan organization key.

19.3.1 The Data Analysis Summary Chart Contents

The analysis chart has five sections. These sections are denoted by the right hand margin numerals in Figure 19-1. The first is an initial conditions section, which identifies the ATARS site, the ATARS program release, and the aircraft state (controller state, equipment, ATARS service state, location, altitude, heading, speed and vertical rate of each aircraft) as defined in the first scan of information. This first section is displayed in two parts, the site name, program release, date and aircraft identities appearing on every page of the form, the aircraft state only appearing once at the beginning of the chart. The second section is a scan-by-scan summary of ATARS messages and the basic conditions causing the generation of these messages. The duration of this section is variable and is defined by the amount of information recorded, i.e., it ends

TABLE 19-4

DATA EXTRACTION SELECTION SETTINGS
FOR PARTICULAR APPLICATIONS

USE	MODE	COMMENTS
Debug	1	Only at test sites due to large amount of data generated
	3,4,5,6,8, 9,10,11,12	Use of these modes will allow a complete history of a scenario to be recorded, as all available information is recorded
Analysis	3,4,8,11,12	These options will generate most of the Data Analysis Summary Chart
Log	2,7	Alarm rate statistics are available from this information

```
***-------------------------------------------------------------------------------------------------------------***
***  site name  -- version        : mn/dd/yr AT hr:mn   --     'aircraft1 id' VS 'aircraft2 id'          PAGE   1 ***
***-------------------------------------------------------------------------------------------------------------***

*=========================================== ENCOUNTER ANALYSIS SUMMARY ============================================*
|                          ID     CONT    EQUIP   SERVC     X        Y       ALT    SPEED     HDG     VRATE          |
|                         AC 1    CTL     EQAB      1      0.0      -.3     3050     170     90.0    -399            |
|                         AC 2    CTL     EQAB      1      1.6      -.5     2050      80     68.0     600            |
*===================================================================================================================*
|SCAN:               |   0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     |
|TIME:               |   0.0    4.7    9.4    14.1   18.8   23.5   28.2   32.9   37.6   42.3   47.0   51.7   56.4   61.1   65.8   |
|AC1 UPLINK:         |          P      P      T      T      T      T      T      DD     DD     DD     DD     DD     DD     DD     |
|AC2 UPLINK:         |          P      P      T      T      T      T      T      DC     DC     DC     DC     DC     DC     DC     |
|CA AC1:             |                                      DD     DD     DD     DD     DD     DD     DD     DD     DD     DD     |
|CA AC2:             |                                      DC     DC     DC     DC     DC     DC     DC     DC     DC     DC     |
|DEL1/DEL2:          |                                1/1   1/1    1/1    1/1    3/3    3/3    3/3    3/3    3/3    3/3    3/3    |
|FAZ/AT AC1/AC2:     | 03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  03/03  |
|AC1 HMS/TURN:       |  0/ 3   0/ 1   0/ 1   0/-3   0/ 0   0/ 1   0/ 0   0/ 0   0/ 1   0/ 1   0/-3   0/-1   0/-1   0/ 3   0/-3  |
|AC2 HMS/TURN:       |  0/ 3   0/ 1   0/-3   0/-1   0/ 0   0/ 1   0/ 1   1/ 2   0/ 2   0/ 2   0/ 2   0/ 0   0/-1   0/-1   0/-1  |
|AC1 OB/RA/TC/TR:    |  0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000  |
|AC2 OB/RA/TC/TR:    |  0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000  |
|CAPD/CHDPD/IPRPD:   | 2/2/0  2/2/0  0/5/0  0/5/0  0/6/0  0/6/0  0/0/0  0/0/0  0/0/0  0/0/0  0/0/0  0/0/0  0/0/0  0/0/0  0/0/0  |
|ICAPLG/MTTPLG:      |  0/0    0/0    0/0    0/0    0/0    0/0    0/0    0/1    0/1    0/1    0/1    0/1    0/1    0/1    0/1  |
*-------------------------------------------------------------------------------------------------------------------*
|SCAN:               |  15     16     17     18     19     20     21                                                  |
|TIME:               |  70.5   75.2   79.9   84.6   89.3   94.0   98.8                                                |
|AC1 UPLINK:         |  DD     P      P      P      P      P      P                                                   |
|AC2 UPLINK:         |  DC     P      P      P      P      P      P                                                   |
|CA  AC1/DEL1:       |  DD                                                                                            |
|CA  AC2/DEL2:       |  DC                                                                                            |
|FAZ/AT AC1/AC2:     | 03/03  03/03  03/03  03/03  03/03  03/03  03/03                                                |
|AC1 HMS/TURN:       |  0/-1   0/-1   0/ 3   0/ 1   0/-3   0/ 3   0/ 1                                               |
|AC2 HMS/TURN:       |  0/-1   0/-1   0/-1   0/-1   0/ 3   0/-3   0/ 3                                               |
|AC1 OB/RA/TC/TR:    |  0000   0000   0000   0000   0000   0000   0000                                               |
|AC2 OB/RA/TC/TR:    |  0000   0000   0000   0000   0000   0000   0000                                               |
|CAPD/CHDPD/IPRPD:   | 1/1/0  1/1/0  1/1/0  1/1/0  1/1/0  1/1/0  1/1/0                                                |
|ICAPLG/MTTPLG:      |  0/0    0/0    0/0    0/0    0/0    0/0    0/0                                                 |
*===================================================================================================================*
| EVENT SUMMARY              | SCAN  ENAT |    DOT      TR_THR   TAU_R     RANGE | TV_THR  TAU_V   REL_ALT   VERT_DIV |   MD     TRANGL |
| PWIPLG SET                 |   0     3  | -0.04266     0.0     53.1    1.6177 |    0.0   67.7   -1145.5   -1018.5 | 0.189    -22   |
| PRWPLG/PPIPLG SET          |   2     3  | -0.03591    45.0     41.5    1.3596 |   45.0   57.4    -982.5   -1026.7 | 0.295    -22   |
| CAPLG SET                  |   2     3  | -0.03591    59.0     41.5    1.3596 |   59.0   57.4    -982.5   -1026.7 | 0.295     -1   |
| RA DETERMINED, RASENT=12   |   4     3  | -0.03900    45.0     30.2    1.0110 |   45.0   53.0    -902.1    -957.2 | 0.251     -1   |
|   EVENT FOLLOW-UP          |   5     3  | -0.30129    45.0     21.5    0.9630 |   45.0   52.1    -789.0    -902.3 | 0.239     -1   |
| CHDPLG/IPPPLG SET          |   6     3  | -0.02283    30.0     16.3    0.8510 |   30.0   50.1    -714.1    -855.2 | 0.216     -1   |
| RA DETERMINED, RASENT= 1   |   7     3  | -0.01948     0.0     10.6    0.7508 |    0.0   47.3    -635.3    -805.4 | 0.004     -1   |
|   EVENT FOLLOW-UP          |   9     3  | -0.01572     0.0     -0.9    0.5935 |    0.0   34.2    -552.4    -969.9 | 0.009     -1   |
| RESOLUTION(S) DROPPED      |  15     3  |  0.00438     0.0     74.4    0.1822 |    0.0 2025.4    -506.3     -15.0 | 0.027     -1   |
|   EVENT FOLLOW-UP          |  16     3  |  0.00810     0.0     29.1    0.3333 |    0.0 2025.4    -506.3     -15.0 | 0.022     -1   |
*===================================================================================================================*
|              |CPA: SLANT =   299 FT;  HOR =    91 FT;  VERT =   283 FT.  SCAN =    13 |                               |
*===================================================================================================================*
```

FIGURE 19-1
DATA ANALYSIS SUMMARY CHART EXAMPLE

```
***--------------------------------------------------------------------------------------------------------------***
***  site name  -- version      : mn/dt/yr AT hr:mn:sc --     'aircraft1 id' VS 'aircraft2 id'             PAGE   2 *** 1
***--------------------------------------------------------------------------------------------------------------***


RASET  =  12          MANTM =  35 SEC    CLIMB RATE AC1 =  700 FPM,  CLIMB RATE AC2 =  800 FPM

*** SEPARATION AT QTIME:
      THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)                 HORIZONTAL PSEP        VERTICAL PSEP

      TL2   CS2   TR2         TL2   CS2   TR2          TL2   CS2   TR2            TL2   CS2   TR2              VMDA    VMDB
TL1  2526  2796  2946    TL1  4643  4796  4885    TL1  3036  3265  3394     TL1  1760  2131  2324     LVL 1   362     389
CS1  2289  2460  2543    CS1  4519  4608  4653    CS1  2843  2982  3051     CS1  1400  1665  1785     LVL 2   859     553
TR1  2271  2307  2277    TR1  4510  4528  4513    TR1  2828  2857  2833     TR1  1370  1430  1380     LVL 3   495     495

*** PSEP MATRICES BEFORE CONVERGENCE CHECKS:

      THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)                 HORIZONTAL PSEP        VERTICAL PSEP

      TL2   CS2   TR2         TL2   CS2   TR2          TL2   CS2   TR2            TL2   CS2   TR2              VMDA    VMDB
TL1  2010  2796  2924    TL1  3842  3841  3839    TL1  2885  3258  3300     TL1  1495  2120  2184     LVL 1    18     256
CS1  1345  1954  2486    CS1  3837  3837  3834    CS1  2477  2846  3051     CS1   128  1407  1785     LVL 2   495     553
TR1  1938  1316  1729    TR1  3833  3833  3830    TR1  2727  2522  2651     TR1  1148   488   953     LVL 3   495     495


*** PSEP MATRICES AFTER CONVERGENCE CHECKS:
```

FIGURE 19-1
DATA ANALYSIS SUMMARY CHART EXAMPLE
(CONTINUED)

```
***-----------------------------------------------------------------------------------------------------------***  -
***  site name  -- version       : mm/dt/yr AT hr:mn:sc --     'aircraft1 id' VS 'aircraft2 id'            PAGE   3 *** 1
***-----------------------------------------------------------------------------------------------------------***  -

      THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)               HORIZONTAL PSEP         VERTICAL PSEP

      TL2   CS2   TR2          TL2   CS2   TR2          TL2   CS2   TR2            TL2   CS2   TR2               VMDA   VMDB
TL1  2010  2796  2924    TL1  3842  3841  3839    TL1  2885  3258  3300      TL1  1485  2120  2184      LVL 1     3    256
CS1  1345  1954  2486    CS1  3837  3837  3834    CS1  2477  2846  3051      CS1   128  1407  1785      LVL 2   495    553
TR1  1938  1316  1729    TR1  3833  3833  3830    TR1  2727  2522  2651      TR1  1148   488   953      LVL 3   495    495


                    C/D       TL/TR      TL/TL      TL/TR C/D  TR/TR C/D  TL/TL C/D  TR/TL C/D  TR/TR      TR/TL

DELIVERABLE          1          1          1          1          1          1          1          0          0

DIR AVAILABLE        1          1          1          1          1          1          1          1          1

COMP W/PRIOR CMDS    1          1          1          1          1          1          1          1          1

PSEP > SEP1          1          1          1          1          1          1          1          1          1

FAR FROM RADAR       0          0          0          0          0          0          0          0          0

HDG SUFFICES         1          0          0          0          0          0          0          0          0

HDG -REV MANVR       0          0          0          0          0          0          0          0          0

HDG BIG PSEP         1          0          0          0          0          0          0          0          0

PST UNCMD/SLO CMD    0          0          0          0          0          0          0          0          0

UNEQUIP BIG HD       0          0          0          0          0          0          0          0          0

NO LEVEL OFF         0          0          0          0          0          0          0          0          0

DETERIORATION        0          0          0          0          0          0          0          0          0

FINAL APPROACH       0          0          0          0          0          0          0          0          0

'PATH' DEPENDENT     1          1          1          0          0          0          0          1          1

PSEP > 2571/ 2574    1          1          0          1          1          1          1          0          0

COMP W/TURN          1          1          0          1          1          0          0          1          0

-- LARGE VMD         1          0          0          1          1          1          1          0          0

-- LARGE HMD         0          0          0          0          0          0          0          0          0

REIN PRIOR CMDS      0          0          0          0          0          0          0          0          0

SPEED CHECK          0          0          0          0          0          0          0          0          0

REIN TURN            0          1          0          1          1          0          0          1          0

BIGGEST PSEP         1          0          0          0          0          0          0          0          0
```

FIGURE 19-1
DATA ANALYSIS SUMMARY CHART EXAMPLE
(CONTINUED)

```
***-----------------------------------------------------------------------------------------------------***
***  site name  -- version       : mn/dt/yr AT hr:mn:sc --     'aircraft1 id' VS 'aircraft2 id'            PAGE   4 *** 1
***-----------------------------------------------------------------------------------------------------***


BASET  =  1          MARTM =  . SEC    CLIMB RATE AC1 =  700 FPM,  CLIMB RATE AC2 =  800 FPM


*** SEPARATION AT QTIMP:
     THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)                 HORIZONTAL PSEP        VERTICAL PSEP

      TL2   CS2   TR2           TL2   CS2   TR2           TL2   CS2   TR2            TL2   CS2   TR2             VMDA   VMDB
TL1  2526  2796  2946     TL1  4643  4796  4885     TL1  3036  3265  3394     TL1  1760  2131  2324     LVL 1   362    389
CS1  2289  2460  2543     CS1  4519  4608  4653     CS1  2843  2982  3051     CS1  1400  1665  1785     LVL 2   859    553
TR1  2271  2307  2277     TR1  4510  4528  4513     TR1  [illegible]128  2857  2833     TR1  1370  1430  1380     LVL 3   495    495

*** PSEP MATRICES BEFORE CONVERGENCE CHECKS:

     THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)                 HORIZONTAL PSEP        VERTICAL PSEP

      TL2   CS2   TR2           TL2   CS2   TR2           TL2   CS2   TR2            TL2   CS2   TR2             VMDA   VMDB
TL1  2010  2796  2924     TL1  3842  3841  3839     TL1  2885  3258  3300     TL1  1485  2120  2184     LVL 1    18    256
CS1  1345  1954  2486     CS1  3837  3837  3834     CS1  2477  2846  3051     CS1   128  1407  1785     LVL 2   495    553
TR1  1938  1316  1729     TR1  3833  3833  3830     TR1  2727  2522  2651     TR1  1148   488   953     LVL 3   495    495


*** PSEP MATRICES AFTER CONVERGENCE CHECKS:
```

FIGURE 19-1
DATA ANALYSIS SUMMARY CHART EXAMPLE
(CONTINUED)

```
***-------------------------------------------------------------------------------------------------------------*** -
***  site name  -- version       : mn/dy/yr AT hr:mn:sc --     'aircraft1 id' VS 'aircraft2 id'              PAGE   5 *** 1
***-------------------------------------------------------------------------------------------------------------*** -

      THREE-DIMENSIONAL (WEIGHTED) PREDICTED SEPARATION (FEET)                  HORIZONTAL PSEP        VERTICAL PSEP

        TL2   CS2   TR2            TL2   CS2   TR2            TL2   CS2   TR2           TL2   CS2   TR2             VMDA   VMDB
TL1    2010  2796  2924     TL1   3842  3841  3839     TL1   2885  3258  3300     TL1   1485  2120  2184     LVL 1      0    256
CS1    1385  1954  2486     CS1   3837  3837  3834     CS1   2477  2846  3051     CS1    128  1407  1785     LVL 2    495    553
TR1    1938  1316  1729     TR1   3833  3833  3830     TR1   2727  2522  2651     TR1   1148   488   953     LVL 3    495    495


                     C/D        TL/TR       TL/TL      TL/TR C/D   TR/TR C/D   TL/TL C/D   TR/TL C/D   TR/TR       TR/TL

DELIVERABLE           1           1           1           1           1           1           1           0           0

DIM AVAILABLE         1           1           1           1           1           1           1           1           1

COMP W/PRIOR CMDS     1           1           1           1           1           1           1           1           1

PSEP > SEP1           1           1           1           1           1           1           1           1           1

FAR FROM RADAR        0           0           0           0           0           0           0           0           0

NEG SUFFICES          1           0           0           0           0           0           0           0           0

NEG ~REV MANVR        0           0           0           0           0           0           0           0           0

NEG BIG PSEP          1           0           0           0           0           0           0           0           0

PST UNCMD/SLO CMD     0           0           0           0           0           0           0           0           0

UNEQUIP BIG FD        0           0           0           0           0           0           0           0           0

NO LEVEL OFF          0           0           0           0           0           0           0           0           0

DETERIORATION         0           0           0           0           0           0           0           0           0

FINAL APPROACH        0           0           0           0           0           0           0           0           0

'PATH' DEPENDENT      1           1           1           0           0           0           0           1           1

PSEP > 2571/ 2574     1           1           0           1           1           1           1           0           0

COMP W/TURN           1           1           0           1           1           0           0           1           0

== LARGE VMD          1           0           0           1           1           1           1           0           0

== LARGE HMD          0           0           0           0           0           0           0           0           0

REIN PRIOR CMDS       0           0           0           0           0           0           0           0           0

SPEED CHECK           0           0           0           0           0           0           0           0           0

REIN TURN             0           1           0           1           1           0           0           1           0

BIGGEST PSEP          1           0           0           0           0           0           0           0           0
```

FIGURE 19-1
DATA ANALYSIS SUMMARY CHART EXAMPLE
(CONCLUDED)

when the extracted information ends for the aircraft(s). The third section lists the value of important parameters at the time of a significant ATARS event, such as calling RAER. The fourth section gives the closest separation occurring during the analysis recording. The final section gives the PSEP and feature array values used by RAER. This section only appears when RAER has been used to generate resolution advisories during the course of the depicted synopsis.

The chart as presented allows for a two aircraft summary. An obstacle avoidance message however involves only one aircraft, and thus generates a chart with only a few entries. Similarly, the mode options selected in extraction will determine the information available to be included in the chart. The details of the five sections of the chart are presented next.

19.3.1.1 Initial Conditions Information

1. System - The system field indicates the origin of the data, i.e., the ATARS site name.

2. ATARS Algorithm Version - This denotes the ATARS release identifier from which data is being extracted.

3. Date and Time - The date and time (Greenwich mean) when data is first recorded. The time used is the start of the sector in which the aircraft appears. TEN is the name by which the time is referenced in the ATARS psuedocode. Either aircraft's time can be used. However, the same aircraft's time must be used throughout the encounter.

4. Aircraft Identification - This field specifies the DABS ID or ATCRBS/Radar code with surveillance file number, the nine bit abbreviated aircraft ID in the State Vector, and the CREFX entry, if any, for each aircraft in this scenario. The CREFX entry is used to establish the correspondence between two data analysis summaries for the same aircraft, both of which originated from separate ATARS sites.

5. Aircraft Control Status - The control status of the aircraft is specified in the State Vector.

6. ATARS Equipage - The equippage of the aircraft is specified in the State Vector. For an aircraft which is both ATARS and BCAS equipped, EQAB is the appropriate label. For ATARS only equipped, EQA is used, for an unequipped aircraft, UNEQ is used.

7. ATARS Service Class - The value of ACLASS is specified in the State Vector.

8. Horizontal Position, Altitude, Heading, Speed, Vertical Rate of Each Aircraft - This information is for the first scan recorded. It is available with the exception of the heading in all extraction modes. The heading can be calculated using the velocity components. The units used will be nautical miles, feet, knots, degrees, as appropriate. In general, parameters referring to the horizontal dimension are in nautical miles or knots, the vertical dimension feet or feet/second. All information related to mensuration throughout the summary chart will have these same units unless explicitly mentioned.

19.3.1.2 Scan-by-Scan Data

1. Time and Scan Number - The scan number used serves as an identifier from one scan's information to the next. As such it will always start with zero and be incremented by one regardless of the elapsed time. The time recorded is the time in seconds (to the nearest tenth) from the start time as specified in item 2 of the initial conditions field.

2. Aircraft Uplink Messages - The resolution advisories calculated in Master Resolution Task as extracted in mode 8 appear in this field. The abbreviations for the advisories are in Table 19-5. Notice that the resolution advisories print positions are one scan after the call to RAER, i.e., at the time the message is displayed in the aircraft.

3. Controller Alert Messages - Each time a controller alert message is generated and extracted under mode 12 for a particular scan, the resolutions specified are to appear in this position. The contents of the DEL field in the controller alert message for each aircraft also appear in the summary sheet. The code used for the advisories is the same as that in aircraft uplink messages. The print position is also delayed by one scan as in item (2) above.

TABLE 19-5
MESSAGE ABBREVIATIONS

MESSAGE	ABBREVIATION
Proximity	P
Threat	T
No Message	Blank
Turn Left	L
Turn Right	R
Climb	C
Descend	D
Don't turn left	NL
Don't turn right	NR
Don't climb	NC
Don't descend	ND
Limit Climb to 500 ft/min	C5
Limit Climb to 1000 ft/min	C1
Limit Climb to 2000 ft/min	C2
Limit Descent to 500 ft/min	D5
Limit Descent to 1000 ft/min	D1
Limit Descent to 2000 ft/min	D2

4. Final Approach Zones and Area Types - The appropriate zone and type for each aircraft is displayed for each scan. These parameters are extracted with options 3 and 4.

5. HMS (Horizontal Maneuver Status)/TURN (Turn Sensing Status) - These are available under any mode option.

6. CAFD/CMDFD/IFRFD, ICAFLG/MTTFLG - The path flags which are as specified with modes 4, 5, or 6. These path parameters should not be confused with the same names used for flags in the Detect Task.

7. OB/RA/TC/TR - This space refers to the terrain/airspace/obstacle avoidance flags extracted in mode 11.

19.3.1.3 Significant Event Summary Data

For each of the following events a group of parameters is recorded which describes the state of the encounter.

1. PWIFLG, FPWFLG, FPIFLG, CAFLG, CMDFLG, IFRFLG - Whenever these flags are set for the first time the parameters listed below are to be included in the encounter analysis summary. The formatting program must check the contents of each scan's dump of modes 4, 5 or 6 for these flag settings. Whenever a flag transitions from not set to set the information is displayed. It is possible for the first scan to have one of the flags set, in which case this scan's information is used.

2. Resolution Advisories Determined (RA SET) - The information for determining if this routine has been called is contained within options 8 or 12. This line may be repeated as often as resolution advisories are generated during the same scan.

3. Resolutions Dropped - This event is determined by examining the CMDFLG flag extracted under options 4, 5, and 6 for each scan appearing in the analysis summary. The transition from a set to a not set condition defines resolution advisories dropped. The scan information may not be available to determine if this condition has occurred (see Section 19.2).

4. EVENT Follow-up - Whenever resolution advisories are determined or a resolution(s) dropped condition occurs, the succeeding scan information is to be displayed. The succeeding scan may or may not be available.

The parameter values appearing with each of the above events are:

a. Scan Number - This is an arbitrary identifier as defined in (1) in the scan-by-scan data section. This is the scan in which the event has occurred.

b. ENAT, DOT (nmi^2/s), TH, TV, RANGE, MD, RZ, VRZ - These parameters are recorded under modes 4 or 5, and their values for the scan in question are to be printed. Notice that the range and miss distance are the square roots of the values extracted. RZ and VRZ must be defined consistently from scan to scan. The Detect Task makes no distinction between AC1 and AC2 from one scan to the next and consequently the sign of the values may alternate, as AC1 alternates with AC2. This is to be corrected in the analysis chart.

c. TH THR, TV THR - Under these headings the appropriate threshold, TCONH(V), TFPWH(V), TCMDH(V), TFIFRH(V), TIFRH(V) for the particular flag set is shown. If both the flags are set which correspond to a line, e.g., CMDFLG/IFRFLG, display the lower of the two thresholds. In the example given IFRFLG would normally have the lower corresponding threshold, TIFRH(V).

d. Track Crossing Angle - The angle between the aircraft headings as calculated from the velocity vectors for the appropriate scan.

19.3.1.4 Separation Summary Content

The closest point of approach shall be the minimum slant range that occurs for all scans represented on the analysis chart for one aircraft pair. The slant ranges can be calculated from the (x, y, z) positions recorded for each aircraft. Appearing with the minimum slant range are the corresponding scan number, and the horizontal and vertical components of the slant range.

19.3.1.5 PSEP and Feature Array Content

For each call to RAER the PSEP matrices and the feature evaluation versus resolution advisory set data array are to be formatted and printed. In the example a "1" indicates that the feature is true for a given advisory set. Note that MANTM, the climb rate for each aircraft and the RASET value are also provided.

19.3.2 Formatting Requirements

For any extraction mode chosen, the system parameters (Appendix A) must also be recorded, once, for any extraction cycle. These parameters must be formatted and printed in a clear and concise manner with each value accompanied by its parameter name.

Similarly, data extracted items which are not part of the encounter analysis must be formatted and printed in a clear concise manner. These data are to be labelled and ordered by aircraft pair and within aircraft pair by scan. It shall be the user's option to determine if these additional data are to be printed (if available) in addition to the data analysis summary chart.

APPENDIX A

SYSTEM CROSS-REFERENCE TABLE

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
A	8	MISCVBL.local	
A	9	TAVBL.calculations	
A	4	TRKVBL.predict	
A	13	TURCON.ac1	
ABBREV	3	TA_PROX.advisory_data	
ABBREV	3	TA_THREAT.advisory_data	
ACAB	3	SVECT.general_values	
ACAT	3	SVECT.general_values	
ACCELC	13	MODELING.values	10.72 ft/s^2
ACCELD	13	MODELING.values	10.72 ft/s^2
ACID	3	CTENTRY.data	
ACIDH	3	CTENTRY.data	
ACIDV	3	CTENTRY.data	
ACID1	3	ELENTRY.identifiers	
ACID2	3	ELENTRY.identifiers	
ACLASS	3	SVECT.general_values	
ACLP	3	SVECT.general_values	
ACONTH	3	PDVBL.miscellaneous	Table 8-2
ADET	3	DETPARM.general_parameters	92.5 s^2
ADOT	8	MISCVBL.local	
AF	8	RAVBL.unc_thresholds	Table 8-3
AFCON	8	CAVBL.thresholds	Table 8-1
AFDET	3	DETPARM.general_parameters	Site-dependent
AFIFR	8	TAVBL.ctl_thresholds	Table 8-4
AFPWI	8	TAVBL.unc_thresholds	Table 8-4
AH	8	MISCVBL.local	
AHI	3	CSCREEN.thresholds	12,000 ft
AIFR	8	RAVBL.ctl_thresholds	Table 8-3
AIRSPACE_NO	3	AIRSPACE.status	
AIRSPACE_TYPE	3	AIRSPACE.adv_data	
ALECT	3	SVECT.times	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
ALECTIM	16	DLMCPARM.change_thresholds	300 s
ALO	3	SYSTEM.tracker	10,000 ft
ALPC	12	MRPARM.res_adv_computation	18,000 ft
ALT	3	ATCRBS_TB.track_data	
ALT	3	ELENTRY.computed_separations	
ALT	6	OBLIST.obstacle_data	
ALT_EXT_INCR	9	TAPARM.msg_format	500 ft
ALT_EXT_LIM	9	TAPARM.msg_format	2000 ft
ALT_TIME_FACT	4	RPTPARM.ztrk_init	7.
ALTDC	10	SEAMPARM.miscellaneous	Site-dependent
ALTITUDE	3	ALEC.adv_data	
ALUH	12	MRPARM.res_adv_computation	29,000 ft
APAIR	12	MRPARM.miscellaneous	2
APOS	3	SYSVAR.antenna	
ARATE	3	SYSVAR.antenna	
ASEP	12	MRVBL.res_adv_thr	
ASEP	13	RAERVBL.res_adv	
ASEPH	12	MRPARM.res_adv_computation	670 ft
ASEPIL	12	MRPARM.res_adv_computation	375 ft
ASEPL	12	MRPARM.res_adv_computation	470 ft
ASEPU	12	MRPARM.res_adv_computation	770 ft
ASSOC	3	SVECT.general_values	
ATARS_EQP	3	TA_PROX.advisory_data	
ATARS_EQP	3	TA_THREAT.advisory_data	
ATBZP	12	MRPARM.res_adv_recomputation	0.8
ATCNMC	3	SYSVAR.flags	
ATCRBS_TRACK_NO	3	ATCRBS_TB.identity	
ATCREF	3	SVECT.pointers	
ATCROR	3	SYSVAR.flags	
ATERN	13	RAERPARM.negative_RA	500 ft
ATIFLG	3	SVECT.flags	
ATSEQ	3	SVECT.general_values	
ATSID	3	PREC.identifiers	
ATSS	3	SVECT.flags	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
AV	8	MISCVBL.local	
AVRZ	13	DOMINOVBL.detection	
AZBIN	6	TAO.misc_variables	
AZP	3	SVECT.horz_tracker_data	
B	13	TURCON.ac1	
BACKUP	3	SYSVAR.failure_info	
BACHRADS	13	RAERPARM.pointers	Pointer
BANKA	13	MODELING.values	20 deg
BCASSL	3	SVECT.general_values	
BCSOFF	8	PATHVBL.local	
BDET	3	DETPARM.general_parameters	0.107 nmi^2
BEARING	3	ATCRBS_TB.track_data	
BEARING	9	TAVBL.calculations	
BELOW1000	13	RADS.read/write_flags	
BELOW1000	12	TRADS.read/write_flags	
BETA_MAX	4	TRKPARM.vert_tracker	0.1
BETA1	4	TRKVBL.vert_tracker	
BIGHWGT	13	RAERPARM.feature_weights	2**4
BIGVWGT	13	RAERPARM.feature_weights	2**5
BLIM	4	TRKVBL.vert_tracker	
BOFFREQ	3	ELENTRY.processing_required	
BSEPNWGT	13	RAERPARM.feature_weights	2**0
BSEPPWGT	13	RAERPARM.feature_weights	2**0
BZP	3	RAPARM.filter_thresholds	
BZP2	3	RAPARM.filter_thresholds	0.9025
CA	13	TURCON.ac1	
CACRD	3	RPALST.ovrhd	
CAFLG	3	ELENTRY.detect_flags	
CAMA	8	MTPARM.cntr_thresholds	1000 ft
CAMCP2	8	MTPARM.cntr_thresholds	0.981
CAMRM2	8	MTPARM.cntr_thresholds	0.00244 nmi^2
CAMR2	8	MTPARM.cntr_thresholds	3.25 nmi^2
CAMSB2	8	MTPARM.cntr_thresholds	0.117
CAMVSQ	8	MTPARM.cntr_thresholds	325 kt^2

AD-A104 148 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-2 FAA-RD-81-45-2 NL

7 OF 7
AD
A104148

END
DATE
FILMED
10 81
DTIC

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
CAREQ	3	SVECT.flags	
CARN	3	RPALST.ovrhd	
CENTR	3	SVECT.flags	
CLIMB	13	PDC_LIST.res_adv	
CLIMB_PERF	3	TA_PROX.advisory_data	
CLIMB_PERF	3	TA_THREAT.advisory_data	
CLM	13	RATE.ac1	
CLMB	13	PRADSVVBL.ac1	
CLOCK_BRG	3	OBSTACLE.adv_data	
CLOCK_BRG	3	TA_PROX.advisory_data	
CLOCK_BRG	3	TA_THREAT.advisory_data	
CLOCK_INCR	9	TAPARM.msg_format	30 deg
CMDED_CMDED	13	RADS.read-only_flags	
CMDED_CMDED	12	TRADS.read-only_flags	
CMDED_UNCMDED	13	RADS.read-only_flags	
CMDED_UNCMDED	12	TRADS.read-only_flags	
CMDFL	3	PREC.ac1	
CMDFLG	3	ELENTRY.detect_flags	
CMAREQ	3	ELENTRY.processing_required	
CNT_DELT	4	TRKPARM.vert_tracker	4.0
CNT_INCR	4	TRKPARM.vert_tracker	10
COAA2	3	AZPARM.arznvb	0.9698
CODE	3	SVECT.general_values	
COMPAT(11,11)	13	LOGIC_TABLES.compatible_res_adv	Table 13-7
COMPATIBLE	5	MARVBL.misc	
COMPATTS(7,6)	13	LOGIC_TABLES.compat_turn_states	Table 13-10
COMPATZD(3,3)	13	LOGIC_TABLES.compat_turn_states	Table 13-11
CONWTWGT	13	RAERPARM.feature_weights	2**6
CONFIDENCE	3	ALEC.adv_data	
CONTROL	3	TA_PROX.advisory_data	
CONTROL	3	TA_THREAT.advisory_data	
CORRECTED	3	ALEC.adv_data	
COSA2	8	MISCVBL.local	
COSP2	8	MTPARM.gnl_thresholds	0.981

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
COURSE	3	TA_PROX.advisory_data	
COURSE	3	TA_THREAT.advisory_data	
COURSE_INCR	9	TAPARM.msg_format	45 deg
CPSID	3	ELENTRY.identifiers	
CRDSCAN	11	CRDPARM.ovrhd	8 s
CTE	3	SVECT.pointers	
CTIME	3	SYSVAR.time	
CTPTR	3	SVECT.pointers	
CTRPTR	3	SYSVAR.failure_info	
CTRTBL(15)	3	SYSTEM.coverage	Site-dependent
CUNC	3	SVECT.flags	
CURM2(3,3)	13	MANGEOM.separation	
CURP2(3,3,3)	13	MANGEOM.separation	
CURV(3)	13	MANGEOM.separation	
DAF	13	DRAVLB.thresholds	
DALT	13	DOMINOVBL.detection	
DBINS	4	TRKVBL.vert_tracker	
DCLIMB	13	PDC_LIST.res_adv	
DCLM	13	RATE.ac1	
DCMDFLG	13	DOMINOVBL.detection	
DDES	13	RATE.ac1	
DDESC	13	PDC_LIST.res_adv	
DDOT(4)	13	DOMINOVBL.detection	
DDSQ	13	DOMINOVBL.detection	
DECAY_FCTR	4	TRKPARM.vert_tracker	0.8 ft/s
DEL	3	RPALST.ac1	
DELAY	13	MODELING.values	10 s
DELFG	3	SVECT.flags	
DELINT	13	MODELING.values	1 s
DELREQ	3	ELENTRY.processing_required	
DELT	4	TRKVBL.vert_tracker	
DELWGT	13	RAERPARM.feature_weights	2**24
DEWAT	13	DOMINOVBL.detection	
DES	13	RATE.ac1	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
DESC	13	PDC_LIST.res_adv	
DETRIMH(5,7,7)	12	MRPARM.logic_tables	Table 12-3
DETRIMV(11,11,3)	12	MRPARM.logic_tables	Table 12-4
DIMAVWGT	13	RAERPARM.feature_weights	2**23
DISCREPANCY	4	TRKPARM.vert_tracker	1.5
DLEFTDRIGHT	13	PDC_LIST.res_adv	
DLOUT	3	SVECT.flags	
DMOD	8	BCSVBL.res	Table 8-1
DMODTA	8	BCSVBL.threat	Table 8-1
DOMCRSE	13	RAERPARM.misc	3.0
DOMNONC	3	SYSTEM.miscellaneous	1
DOMSCANS	13	RAERPARM.misc	4.0
DOMSRCH	13	RAERPARM.misc	3.0
DOMVALUE	13	RADS.other-info	
DOMVALUE	12	TRADS.other-info	
DOM1WGT	13	RAERPARM.feature_weights	2**17
DONEBOTH	8	PATHVBL.local	
DOT	3	ELENTRY.computed_separations	
DOT	13	MODVBL.relative_geometry	
DOTP	9	TAVBL.calculations	
DOTTH	3	DETPARM.general_parameters	0.00278 nmi²/s
DRANGE2	13	DOMINOVBL.detection	
DRATS	3	SVECT.flags	
DRCMD2	13	DRAVLB.thresholds	
DRSUR	3	SVECT.flags	
DRZ	13	DOMINOVBL.detection	
DS	4	TRKVBL.predict	
DSC	13	PRADSVVBL.ac1	
DSQ	8	MISCVBL.local	
DT	6	ACUPVBL.times	
DT	3	SYSTEM.ztrack	4.7 s
DTCMDH	13	DRAVLB.thresholds	
DTCMDV	13	DRAVLB.thresholds	
DTH	13	DOMINOVBL.detection	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
DTL	6	ACUPVBL.times	
DTV	13	DOMINOVBL.detection	
DVDPT	3	RAPARM.filter_thresholds	30 s
DVPZ	13	DOMINOVBL.detection	
DZM	4	TRKVBL.vert_tracker	
DZ10	4	TRKVBL.vert_tracker	
D2	4	TRKVBL.smoothing	
D2TH	4	TRKVBL.smoothing	
EFFHRA(7,7)	12	MRPARM.logic_tables	Table 12-5
EFFVRA(13,13)	12	MRPARM.logic_tables	Table 12-6
EHMAN	3	PREC.ac1	
ELENTRY	12	MRVBL.pointer	
ELENTRY	13	RAERVBL.pointers	
ENAT	3	ELENTRY.geographic_dependent	
ENAT	13	PDC_LIST.detection	
END	3	AIRSPACE.status	
END	3	ATCRBS_TB.identity	
END	3	OBSTACLE.status	
END	3	TA_PROX.identity	
END	3	TA_THREAT.identity	
END	3	TERRAIN.status	
EVMAN	3	PREC.ac1	
EXFLG	3	SVECT.flags	
EXITLOOP	8	PATHVBL.local	
EXVEL	13	RAERPARM.domino	600 kt
FACHRADS	13	RAERPARM.pointers	Pointer
FAILED	3	SYSVAR.failure_info	
FARRANGT	13	RAERPARM.feature_weights	2**16
FAZ	3	SVECT.general_values	
FAZWGT	13	RAERPARM.feature_weights	2**9
FCTE	3	CTHEAD.maintenance	
FEATBITS(25)	13	RADS.other-info	
FEATBITS(25)	12	TRADS.other-info	
FESTAB	4	TRKPARM.trk_quality	6

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
FILE	3	SVECT.general_values	
FILTFAIL	8	PATHVBL.local	
FINE_BRG	3	TA_PROX.advisory_data	
FINE_BRG	3	TA_THREAT.advisory_data	
FINE_BRG_INCR	9	TAPARM.msg_format	3.75 deg
FINE_HDG	3	TA_THREAT.advisory_data	
FINE_HDG_INCR	9	TAPARM.msg_format	2.8125 deg
FIRHE	3	SVECT.horz_tracker_data	
FIRHI	3	SVECT.horz_tracker_data	
FIRHZ	3	SVECT.vert_tracker_data	
FIRHZ_MAX	4	TRKPARM.vert_tracker	9.0
FIRHZR	3	SVECT.vert_tracker_data	
FIRHZR_INCR	4	TRKPARM.vert_tracker	0.6
FIRHZR_INIT	4	RPTPARM.ztrk_init	5
FIRHZR_MAX	4	TRKPARM.vert_tracker	10.0
FIRHZR_MIN	4	TRKPARM.vert_tracker	2.0
FPIFLG	3	ELENTRY.detect_flags	
FPWFLG	3	ELENTRY.detect_flags	
FSTUNCZD	13	RAERVBL.res_adv	
FTAT	3	AIRSPACE.status	
FTAT	3	OBSTACLE.status	
FTAT	3	TERRAIN.status	
FUCSCWGT	13	RAERPARM.feature_weights	2**13
G	13	MODELING.values	32.16 ft/s²
GEOG	3	SVECT.general_values	
GOTHT	8	PATHVBL.local	
GRND_SPEED	3	TA_PROX.advisory_data	
GRND_SPEED	3	TA_THREAT.advisory_data	
HALFSEC	15	USIPARM.values	8
HDOFF	3	PREC.identifiers	
HEADING	9	TAVBL.calculations	
HIT	4	TRKVBL.logic_path	
HMAN	3	CTENTRY.data	
HMAND	3	CTENTRY.data	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
HMD	3	TA_THREAT.advisory_data	
HMD2(3,3)	13	PSMAT.minimums	
HMD2I	13	DELGEOM.minsep	
HMS	3	SVECT.general_values	
HORIZ	13	RADS.read-only_flags	
HORIZ	12	TRADS.read-only_flags	
HPROX	8	PATHVBL.local	
HUBFLG	3	SVECT.flags	
HUBRAD	3	SYSTEM.miscellaneous	10 nmi
H1	8	BCSVBL.res	Table 8-1
H1	13	RADS.advisory_components	
H1	12	TRADS.advisory_components	
H1TA	8	BCSVBL.threat	Table 8-1
H2	13	RADS.advisory_components	
H2	12	TRADS.advisory_components	
ICAFLG	3	ELENTRY.detect_flags	
ID	3	RPALST.ac1	
IDENTIFIER	3	AIRSPACE.adv_data	
IFRFLG	3	ELENTRY.detect_flags	
INIT	4	TRKVBL.smoothing	
IND	3	SVECT.general_values	
INDEX1	13	RADS.sep_matrix_indices	
INDEX1	12	TRADS.sep_matrix_indices	
INDEX2	13	RADS.sep_matrix_indices	
INDEX2	12	TRADS.sep_matrix_indices	
INDEX3	13	RADS.sep_matrix_indices	
INDEX3	12	TRADS.sep_matrix_indices	
INFAZ2	13	DOMINOVBL.detection	
INFAZ2	8	ELVBL.local	
INTR	3	PREC.ac1	
INTRAC	13	PDC_LIST.pointer	
INXFL	3	SVECT.flags	
INZONE	6	ACUPVBL.flags	
ISGN	4	TRKVBL.vert_tracker	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
LARGET	9	TAPARM.ranking	1000 s
LEFT	13	PDC_LIST.res_adv	
LEFTCLIMB	13	PDC_LIST.res_adv	
LEFTDESC	13	PDC_LIST.res_adv	
LEVEL_TIME	4	TRKPARM.vert_tracker	99.0
LFT	13	PRADSVVBL.ac1	
LFTCLMB	13	PRADSVVBL.ac1	
LFTDSC	13	PRADSVVBL.ac1	
LIL_BIT	4	TRKPARM.vert_tracker	1.0
LOCAL_ID	3	SYSVAR.general	
LOFL	3	SVECT.flags	
LOT	3	SVECT.vert_tracker_data	
LOT_SCALE	4	TRKPARM.vert_tracker	0.4
LSTPTR	6	NEWACVBL.pointers	
MANTM	13	MODVBL.miscellaneous	
MAPPTR	3	SYSVAR.failure_info	
MAPTBL(15)	3	SYSTEM.coverage	Site-dependent
MASTER	3	SYSVAR.failure_info	
MASTRTBL(15)	3	SYSTEM.coverage	Site-dependent
MATCHED	9	TAVBL.identity	
MATPTR	13	RADS.sep_matrix_indices	
MATPTR	12	TRADS.sep_matrix_indices	
MAXAF	13	RAERPARM.domino	750.0 ft
MAXTLI	13	RAERPARM.domino	60.0 s
MAXTLV	13	RAERPARM.domino	60.0 s
MAXVALUE	13	DOMINOVBL.detection	
MCFLG	3	SVECT.flags	
MCTA	11	CRDPARM.ovrhd	3
MDCON2	8	CAVBL.thresholds	Table 8-1
MDFPI2	9	TAVBL.ctl_thresholds	Table 8-4
MDFPW2	8	TAVBL.unc_thresholds	Table 8-4
MDHM	13	RAERVBL.res_adv	
MDHMSQ	13	RAERPARM.features	0.489 nmi²
MDHSQ	13	RAERPARM.features	0.083 nmi²

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
MDTHM	12	MRVBL.res_adv_thr	
MDTHM	13	RAERVBL.neg_res_adv	
MDTHMSQ	3	RESADV.thresholds	0.831 nmi²
MDTHSQ	3	RESADV.thresholds	0.25 nmi²
MD2	3	ELENTRY.computed_separations	
MD2NA	8	NATAPARM.nathrs	4.0 nmi
MISS_FCTR	4	TRKPARM.vert_tracker	0.6
MODEL(3)	13	PATH.ac1	
MOPVRADS	13	RAERPARM.pointers	Pointer
MRATE	13	RAERPARM.negative_RA	6.67 ft/s
MRNCAP	12	MRVBL.logic_path	
MRNCAP	13	RAERVBL.logic_path	
MSMVRADS	13	RAERPARM.pointers	Pointer
MT_DETECTED	8	PATHVBL.local	
MTLL	13	MODELING.values	20 s
MTSC	13	MODELING.values	60 s
MTTA	8	MTPARM.gnl_thresholds	1000 ft
MTTFLG	3	ELENTRY.detect_flags	
MTTRM2	8	MTPARM.gnl_thresholds	0.00244 nmi²
MTTR2	8	MTPARM.gnl_thresholds	3.25 nmi²
MTTSB2	8	MTPARM.gnl_thresholds	0.117
MTTVSQ	8	MTPARM.gnl_thresholds	325 kt²
MTUL	13	MODELING.values	100 s
MULT	8	ELVBL.local	
MULT	13	PDC_LIST.detection	
MULTH	3	CTENTRY.data	
MULTV	3	CTENTRY.data	
MVDONE	3	PREC.model_validation	
MVRAIT	3	PREC.model_validation	
MVT	3	PREC.ac1	
MVVRZ	3	PREC.model_validation	
MVZDP	12	MRPARM.res_adv_recomputation	0.2
MVZDM	12	MRPARM.res_adv_recomputation	300 ft/min
NAC	3	CTHEAD.data	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
NCLMB	13	PRADSVVBL.ac1	
NCON	3	CTENTRY.data	
NCTA	11	CRDPARM.ovrhd	5
NDNRMWGT	13	RAERPARM.feature_weights	2**14
NDONWGT	13	RAERPARM.feature_weights	2**18
NDSC	13	PRADSVVBL.ac1	
NEAR1	15	USIPARM.values	2
NEAR2	15	USIPARM.values	14
NEGATIVE	13	RADS.read/write_flags	
NEGATIVE	12	TRADS.read/write_flags	
NEGDIV	13	RAERVBL.neg_res_adv	
NEGSPWGT	13	RAERPARM.feature_weights	2**15
NEW_HMS	4	TRKVBL.smoothing	
NEW_TM	4	TRKVBL.predict	
NEXTA	3	SVECT.pointers	
NEXTCT	3	CTHEAD.maintenance	
NEXTO	6	OBLIST.obstacle_data	
NEXTS	3	SVECT.pointers	
NEXTX	3	SVECT.pointers	
NLPTNRGT	13	PRADSVVBL.ac1	
NO_CONT	7	CSVBL.xclud_types	
NO_NONC	7	CSVBL.xclud_types	
NOCA	8	PATHVBL.local	
NOI	3	AZPARM.counts	Table 8-6
NOII	3	AZPARM.counts	Table 8-6
NOLEVWGT	13	RAERPARM.feature_weights	2**11
NORES	8	PATHVBL.local	
NOTHREAT	8	PATHVBL.local	
NOZ1	3	AZPARM.counts	Table 8-7
NOZ2	3	AZPARM.counts	Table 8-7
NPRAABS	13	RAERVBL.res_adv	
NRESPWGT	13	RAERPARM.feature_weights	2**10
NSIGNP	6	NEWACVBL.signp	
NSVDAT	13	RAERPARM.negative_RA	200 ft

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
NSVDPT	13	RAERPARM.negative_RA	30 s
NULLFG	3	SVECT.flags	
NUMBER	6	OBLIST.obstacle_data	
NUMPRA	13	DOMINOVBL.detection	
NXCTE	3	CTENTRY.maintenance	
NXTAC	13	DOMINOVBL.coarse_screen	
NXTADV	13	RADS.pointers	
NXTADV	12	TRADS.pointers	
NXTINTR	13	PDC_LIST.pointer	
NXTPR	3	PREC.maintenance	
NXTPWI	3	AIRSPACE.maintenance_info	
NXTPWI	3	ALEC.maintenance_info	
NXTPWI	3	ATCRBS_TB.maintenance_info	
NXTPWI	3	OBSTACLE.maintenance_info	
NXTPWI	3	TA_PROX.maintenance_info	
NXTPWI	3	TA_THREAT.maintenance_info	
NXTPWI	3	TERRAIN.maintenance_info	
O_ID	10	SEANVBL.miscellaneous	
OALRT	6	TAO.misc_variables	
OBALT	6	TAOPARM.general_values	3000 ft
OBJ_AC	3	ATCRBS_TB.identity	
OBJ_AC	3	TA_PROX.identity	
OBJ_AC	3	TA_THREAT.identity	
OBJECT	9	TAVBL.identity	
OBSTACLE_NO	3	OBSTACLE.status	
OBXCK	6	TAOPARM.general_values	2000 ft
OBYCK	6	TAOPARM.general_values	2000 ft
OBZCK	6	TAOPARM.general_values	500 ft
OLD_TYPE	3	TA_PROX.identity	
OLD_TYPE	3	TA_THREAT.identity	
ONEXRATE	4	TRKPARM.vert_tracker	5.0 ft/s
OSCFL	3	SVECT.flags	
OSHMAN	12	MRVBL.other_site	
OSHMAN1	13	RAERVBL.res_adv	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
OSHMAN2	13	RAERVBL.res_adv	
OSVMAN	12	MRVBL.other_site	
OSVMAN1	13	RAERVBL.res_adv	
OSVMAN2	13	RAERVBL.res_adv	
OTHSTWGT	13	RAERPARM.feature_weights	2**20
OWN_DELTA_HDG	16	DLMCPARM.change_thresholds	15 deg
OWN_REQD	16	DLMCVBL.miscellaneous	
OWNHDG	3	SVECT.general_values	
OWNID	3	SYSTEM.miscellaneous	Site-dependent
OWNT	3	SVECT.times	
OWNTRN	3	SVECT.general_values	
PAC	3	PREC.ac1	
PART_SCAN	4	TRKPARM.vert_tracker	0.8
PF_FAILED	8	PATHVBL.local	
PHMAN	3	PREC.ac1	
PHRA1	13	PREVIOUS.advisories	
PHRA2	13	PREVIOUS.advisories	
PIPR	3	PREC.general_values	
PLIST	3	CTHEAD.data	
PMD	3	PREC.general_values	
POSCMD	3	PREC.general_values	
PRCONT	13	DOMINOVBL.detection	
PRCONT	8	ELVBL.local	
PREC	12	MRVBL.pointer	
PREC	13	RAERVBL.pointers	
PREQ	13	DOMINOVBL.detection	
PREQ	8	ELVBL.local	
PREVCT	3	CTHEAD.maintenance	
PREVX	3	SVECT.pointers	
PROXNO	16	DLMCVBL.miscellaneous	
PRVPWI	3	AIRSPACE.maintenance_info	
PRVPWI	3	ALEC.maintenance_info	
PRVPWI	3	ATCRBS_TB.maintenance_info	
PRVPWI	3	OBSTACLE.maintenance_info	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
PRVPWI	3	TA_PROX.maintenance_info	
PRVPWI	3	TA_THREAT.maintenance_info	
PRVPWI	3	TERRAIN.maintenance_info	
PSEPSQ	12	MRVBL.other_site	
PSEP1WGT	13	RAERPARM.feature_weights	2**21
PSEP2(3,3,3)	13	PSMAT.minimums	
PSEP2I	13	DELGEOM.minsep	
PSEP2WGT	13	RAERPARM.feature_weights	2**7
PSTAT	3	SVECT.flags	
PVMAN	3	PREC.ac1	
PVMD	3	PREC.general_values	
PVRA1	13	PREVIOUS.advisories	
PVRA2	13	PREVIOUS.advisories	
PWIFLG	3	ELENTRY.detect_flags	
PWISP	3	PREC.general_values	
PWPTR	3	SVECT.pointers	
Q	3	SYSTEM.ztrack	100 ft
QSEP2(3,3,3)	13	PSMAT.snapshot	
QSIGN	4	TRKVBL.vert_tracker	
QTIME	13	MODELING.values	9.4 s
R	8	MISCVBL.local	
RADS	13	RAERVBL.pointers	
RADSPTR	12	MRVBL.pointer	
RADSPTR	13	RAERVBL.pointers	
RALRT	6	TAO.misc_variables	
RANGE	3	ATCRBS_TB.track_data	
RANGE	3	OBSTACLE.adv_data	
RANGE	3	TA_PROX.advisory_data	
RANGE	3	TA_THREAT.advisory_data	
RANGE_RATE	3	ATCRBS_TB.track_data	
RANGE_WEIGHTED	3	TA_PROX.rank_data	
RANGE_WEIGHTED	3	TA_THREAT.rank_data	
RANGE2	3	ELENTRY.computed_separations	
RANKTYP	3	TA_PROX.rank_data	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
RANKTYP	3	TA_THREAT.rank_data	
RAPP1	13	RAERVBL.pointers	
RAPP2	13	RAERVBL.pointers	
RAPROV	3	ELENTRY.processing_required	
RAREQ	3	ELENTRY.processing_required	
RASELECT	12	HRVBL.logic_path	
RASELECT	13	RAERVBL.res_adv	
RATE_FACT	4	TRKPARM.vert_tracker	2.0
RATESMOOTH	4	TRKPARM.vert_tracker	0.1
RCMD2	8	RAVBL.unc_thresholds	Table 8-3
RCONTH	3	PDVBL.miscellaneous	Table 8-2
RCON2	8	CAVBL.thresholds	Table 8-1
RD	8	MISCVBL.local	
RDET	3	DETPARM.general_parameters	Site-dependent
RDIST	3	AZPARM.arznvb	83.3 nmi
RDISTR	13	RAERPARM.features	90 nmi
RDREQ	3	ELENTRY.processing_required	
RDTA	8	MISCVBL.local	
RDTEMP	8	MISCVBL.local	
RDTHR	8	BCSVBL.res	0.00167 nmi/s
RDTHRTA	8	BCSVBL.threat	0.004 nmi²/s
RECALC	12	HRVBL.logic_path	
RECFLG	4	TRKVBL.predict	
REINF(9,11)	13	LOGIC_TABLES.reinf_res_adv	Table 13-8
REINTWGT	13	RAERPARM.feature_weights	2**1
REL_ALT	3	OBSTACLE.adv_data	
REL_ALT	3	TA_PROX.advisory_data	
REL_ALT	3	TA_THREAT.advisory_data	
REL_ALT	3	TERRAIN.adv_data	
REL_ALT_EXT	3	TA_THREAT.advisory_data	
REMFLG	3	CTENTRY.data	
REMRAR	3	SVECT.general_values	
REPORT	3	SVECT.data_block	
REPRAWGT	13	RAERPARM.feature_weights	2**3

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
RES	3	RPALST.ac1	
RESP	10	SEAMVBL.miscellaneous	
RESSENT	16	DLMCVBL.miscellaneous	
RETRAR	3	SVECT.general_values	
RPIFR2	8	TAVBL.ctl_thresholds	Table 8-4
RPPWI2	8	TAVBL.unc_thresholds	Table 8-4
RGT	13	PRADSVVBL.ac1	
RGTCLMB	13	PRADSVVBL.ac1	
RGTDSC	13	PRADSVVBL.ac1	
RHOP	3	SVECT.horz_tracker_data	
RIFR2	8	RAVBL.ctl_thresholds	Table 8-3
RIGHT	13	PDC_LIST.res_adv	
RIGHTCLIMB	13	PDC_LIST.res_adv	
RIGHTDESC	13	PDC_LIST.res_adv	
RMAX	7	CSVBL.bounds	
RMAX	13	DOMINOVBL.coarse_screen	
RMAXH	3	CSCREEN.distances	21.2 nmi
RMAXI	3	CSCREEN.distances	9.2 nmi
RMAXV	3	CSCREEN.distances	6.2 nmi
RMFL	3	SVECT.flags	
RW_TIME_OUT	11	CRDPARM.ovrhd	6 s
RWKTAU	9	TAVBL.calculations	
RPMIN	8	PAPARM.thresholds	4.0 nmi^2
RPTRK	4	RPTVBL.logic_path	
RPWI	3	CSCREEN.distances	4. nmi
RRREJF	4	TRKVBL.logic_path	
RSPND1	13	RAERVBL.res_adv	
RSPND2	13	RAERVBL.res_adv	
RST	8	MISCVBL.local	
RTHETA	8	BCSVBL.threat	Table 8-1
RX	8	MISCVBL.local	
RX	13	MODVBL.relative_geometry	
RX	9	TAVBL.calculations	
RXP	9	TAVBL.calculations	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
RXVS	8	MISCVBL.local	
RY	8	MISCVBL.local	
RY	13	MODVBL.relative_geometry	
RY	9	TAVBL.calculations	
RYP	9	TAVBL.calculations	
RZ	8	MISCVBL.local	
RZ	13	MODVBL.relative_geometry	
RZP	9	TAVBL.calculations	
R2NA	8	MATAPARM.nathrs	4.0 nmi
S	4	TRKVBL.smoothing	
SA	13	TURCON.ac1	
SACHRADS	13	RAERPARM.pointers	Pointer
SCAN_FACTOR	4	TRKPARM.vert_tracker	0.05
SCANT	3	SYSTEM.miscellaneous	4.7 s
SEAM	3	CTHEAD.data	
SECTID	3	PREC.identifiers	
SEND	3	PREC.ac1	
SENT	3	AIRSPACE.maintenance_info	
SENT	3	ALEC.maintenance_info	
SENT	3	ATCRBS_TB.maintenance_info	
SENT	3	OBSTACLE.maintenance_info	
SENT	3	TA_PROX.maintenance_info	
SENT	3	TA_THREAT.maintenance_info	
SENT	3	TERRAIN.maintenance_info	
SEP1	3	RESADV.thresholds	0.0271 nmi^2
SEP2AP	13	RAERPARM.features	0.67
SHIFT_FACT	4	TRKPARM.vert_tracker	64
SINB2	8	MISCVBL.local	
SINGLE	13	RADS.read-only_flags	
SINGLE	12	TRADS.read-only_flags	
SLREPS	3	SVECT.general_values	
SMPR	3	SVECT.flags	
SNGDIM	12	MRVBL.logic_path	
SNGDIM	13	RAERVBL.logic_path	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
SNGLDWGT	13	RAERPARM.feature_weights	2**8
SOURCE	3	RPALST.ovrhd	
SPDCKWGT	13	RAERPARM.feature_weights	2**2
SPIDFG	3	SVECT.flags	
SPLO2	3	SYSTEM.tracker	(240 kt)²
SPRO	3	SVECT.flags	
SQLO	3	SVECT.flags	
SQMAP	3	SYSTEM.coverage	Site-dependent
SR_MAG	4	TRKPARM.vert_tracker	0.2
SR_THRESH	4	TRKPARM.vert_tracker	1.3
SRGAIN	4	TRKPARM.vert_tracker	0.5
SRVMSK	3	SVECT.flags	
SSL	8	ELVBL.local	
START	7	CSVBL.starting_loc	
STATMSG	3	SYSVAR.flags	
STKPTR	3	SVECT.pointers	
SUBFIELDNO	16	DLMCVBL.miscellaneous	
SUBJECT	9	TAVBL.identity	
SUCNT	3	SVECT.vert_tracker_data	
SUMRES	3	SVECT.vert_tracker_data	
SVSID	3	SVECT.general_values	
TACID	12	MRVBL.pointer	
TALRT	6	TAO.misc_variables	
TAREQ	3	ELENTRY.processing_required	
TATSN	6	ACUPVBL.times	
TAU	3	TA_PROX.rank_data	
TAU	3	TA_THREAT.rank_data	
TAUR	8	MISCVBL.local	
TCADEL	13	MODELING.values	17 s
TCALRT	6	TAO.misc_variables	
TCMDH	8	RAVBL.unc_thresholds	Table 8-3
TCMDV	8	RAVBL.unc_thresholds	Table 8-3
TCONH	8	CAVBL.thresholds	
TCONV	8	CAVBL.thresholds	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
TCUR	4	TRKVBL.vert_tracker	
TD	3	SVECT.times	
TDDS	4	TRKPARM.trk_quality	0.1 s
TDDS	4	TRKVBL.predict	
TDROP	4	TRKPARM.trk_quality	19.0 s
TEMP_TYPE	9	TAVBL.identity	
TEMPTR	6	ACUPVBL.pointers	
TEN	6	ACUPVBL.times	
TERALT	3	SVECT.general_values	
TEROBWGT	13	RAERPARM.feature_weights	2**19
TEST	4	TRKVBL.vert_tracker	
TEST_ID	10	SEAMVBL.miscellaneous	
TEST_THRSH	4	TRKPARM.vert_tracker	100.0 ft
TFIFRH	8	TAVBL.ctl_thresholds	Table 8-4
TFIFRV	8	TAVBL.ctl_thresholds	Table 8-4
TFPWIH	8	TAVBL.unc_thresholds	Table 8-4
TFPWIV	8	TAVBL.unc_thresholds	Table 8-4
TH	3	ELENTRY.computed_times	
THMS	4	TRKPARM.trk_quality	5.0 s
THMS	4	TRKVBL.predict	
THNA	8	NATAPARM.nathrs	30 s
THTRU	8	MISCVBL.local	
TH1	4	TRKVBL.smoothing	
TH2	4	TRKVBL.smoothing	
TIFRH	8	RAVBL.ctl_thresholds	Table 8-3
TIFRV	8	RAVBL.ctl_thresholds	Table 8-3
TIME	3	RPALST.ovrhd	
TIMINT	13	MODELING.values	2.35 s
TLA	7	CSVBL.bounds	
TLA	8	MISCVBL.local	
TLD	13	DOMINOVBL.coarse_screen	
TLI	3	CSCREEN.times	75 s
TLPSQ	8	PAPARM.thresholds	900 s²
TLUPD	3	SVECT.times	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
TLV	3	CSCREEN.times	75 s
TM	8	MISCVBL.local	
TM	3	SVECT.times	
TMP	3	SVECT.times	
TMPTR2	6	ACUPVBL.pointers	
TMPTR2	6	NEWACVBL.pointers	
TMR	3	SVECT.times	
TMZ	3	SVECT.times	
TNDEX	4	TRKVBL.vert_tracker	
TNVRAM	13	MODELING.values	20 s
TPREC	12	MRVBL.pointer	
TPREV	4	TRKVBL.vert_tracker	
TRACK_NO	3	TA_PROX.identity	
TRACK_NO	3	TA_THREAT.identity	
TRADS	13	RAERVBL.pointers	
TRALT	6	TAOPARM.general_values	5000 ft
TRANS_FACTOR	4	TRKPARM.vert_tracker	1.2
TRATIO	13	RAERVBL.res_adv	
TRECOM	12	MRPARM.res_adv_recomputation	19 s
TRHTM	6	TAOPARM.general_values	60 s
TRKID	3	PREC.ac1	
TRTHR	8	BCSVBL.res	Table 8-1
TRTHRTA	8	BCSVBL.threat	Table 8-1
TRTHU	13	RAERVBL.neg_res_adv	
TRTRU	8	MISCVBL.local	
TSCMD	12	MRPARM.res_adv_recomputation	10 s
TSEPSQ	8	NATAPARM.nathrs	900 m^2
TSREJF	4	TRKVBL.logic_path	
TSTART	3	PREC.general_values	
TTM	3	RAPARM.times	1
TTPRAL	3	SVECT.times	
TURN	3	SVECT.general_values	
TURN	3	TA_THREAT.advisory_data	
TURN	4	TRKVBL.smoothing	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
TURNA1	13	MODELING.values	180 deg
TURNA2	13	MODELING.values	270 deg
TV	3	ELENTRY.computed_times	
TVALID	12	MRPARM.miscellaneous	2.5
TVALUE	13	RAERVBL.res_adv	
TVERT	13	RAERVBL.res_adv	
TVIOL	6	TAO.misc_variables	
TVMD	8	MISCVBL.local	
TVRULE	13	RAERPARM.misc	8.0
TVTHR	8	BCSVBL.res	Table 8-1
TVTHRTA	8	BCSVBL.threat	Table 8-1
TV1	13	RAERPARM.features	8 s
TV2	13	RAERPARM.features	16 s
TWARN	3	PDVBL.miscellaneous	Table 8-2
TXTH	13	RAERVBL.res_adv	
TXTH1	13	RAERPARM.features	60 deg
TXTH2	13	RAERPARM.features	120 deg
TYPE	3	SVECT.general_values	
TZ1	8	MISCVBL.local	
TZ2	8	MISCVBL.local	
UCLVRWGT	13	RAERPARM.feature_weights	2**12
UNCMDED_CMDED	13	RADS.read-only_flags	
UNCMDED_CMDED	12	TRADS.read-only_flags	
UNMANWGT	13	RAERPARM.feature_weights	2**22
UPNES	3	SVECT.pointers	
UUIND	8	ELVBL.local	
VALUE	13	RADS.other-info	
VALUE	12	TRADS.other-info	
VERT	13	RADS.read-only_flags	
VERT	12	TRADS.read-only_flags	
VERT_RATE	3	ATCRBS_TB.track_data	
VERT_SPD	3	TA_THREAT.advisory_data	
VERTRA1	13	RAERVBL.res_adv	
VERTRA2	13	RAERVBL.res_adv	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
VFASTSQ	13	RAERPARM.features	0.0025 (nmi/s)^2
VMAN	3	CTENTRY.data	
VMAND	3	CTENTRY.data	
VMDA(3)	13	PSMAT.minimums	
VMDAI	13	DELGEOM.minsep	
VMDB(3)	13	PSMAT.minimums	
VMDBI	13	DELGEOM.minsep	
VMDTH	3	DETPARM.general_parameters	
VPCS	3	CSCREEN.thresholds	2000 ft
VPROX	8	PATHVBL.local	
VP1	8	PAPARM.thresholds	2000 ft
VRAP	13	NVGEOM.prevert	
VRAT	8	ELVBL.local	
VRATC	3	THRSPARM.ratios	2.25
VRATIO	13	RAERPARM.features	2.25
VRATTH	3	THRSPARM.ratios	2.25
VRTH2	13	MODELING.values	(10 kt)^2
VRX	8	MISCVBL.local	
VRX	13	MODVBL.relative_geometry	
VRY	8	MISCVBL.local	
VRY	13	MODVBL.relative_geometry	
VRZ	8	MISCVBL.local	
VRZ	13	MODVBL.relative_geometry	
VRZA	8	MISCVBL.local	
VRZCON	3	THRSPARM.ratios	-300 ft/min
VRZTH	3	DETPARM.general_parameters	15 ft/min
VR2	8	MISCVBL.local	
VR2	13	MODVBL.relative_geometry	
VSLOWSQ	13	RAERPARM.features	0.00111 (nmi/s)^2
VSQ	3	SVECT.horz_tracker_data	
VTHSQ	13	MODELING.values	(150 kt)^2
VWEIGHT	3	SYSTEM.miscellaneous	5.0
V1	13	RADS.advisory_components	
V1	12	TRADS.advisory_components	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
V1000	13	MODELING.values	16.67 ft/s
V2	13	RADS.advisory_components	
V2	12	TRADS.advisory_components	
V2000	13	MODELING.values	33.33 ft/s
V500	13	MODELING.values	8.33 ft/s
W	4	TRKVBL.smoothing	
X	13	DELGEOM.horl	
X	13	MANGEOM.horl(3)	
X	6	OBLIST.obstacle_data	
X	3	SVECT.horz_tracker_data	
X(3,4)	13	RAPP_TABLE.positions	
XA	4	TRKVBL.smoothing	
XD	13	DELGEOM.horl	
XD	13	MANGEOM.horl(3)	
XD	3	SVECT.horz_tracker_data	
XD(3,4)	13	RAPP_TABLE.velocities	
XDE	3	SVECT.horz_tracker_data	
XDEN	4	TRKVBL.smoothing	
XDI	3	SVECT.horz_tracker_data	
XDIN	4	TRKVBL.smoothing	
XDPRJ(4)	3	SVECT.domino_obj_proj	
XL	7	CSVBL.limits	
XL	13	DOMINOVBL.coarse_screen	
XLEVEL	4	TRKPARM.vert_tracker	2.5
XMAX	13	DOMINOVBL.coarse_screen	
XMIN	13	DOMINOVBL.coarse_screen	
XP	7	CSVBL.predictions	
XP	3	SVECT.horz_tracker_data	
XPI	3	SVECT.horz_tracker_data	
XPR(9)	13	DOMINOVBL.coarse_screen	
XPRJ(4)	3	SVECT.domino_obj_proj	
XS	4	TRKVBL.smoothing	
XSI	4	TRKVBL.smoothing	
XSP	3	CSCREEN.thresholds	5 nmi

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
XTONORM	4	TRKPARM.vert_tracker	22.0 s
XTRARESID	4	TRKPARM.vert_tracker	0.7
XU	7	CSVBL.limits	
XU	13	DOMINOVBL.coarse_screen	
XUPFL	3	SVECT.flags	
XVEL	13	RAERPARM.domino	240 kt
Y	13	DELGEOM.hor1	
Y	13	MANGEOM.hor1(3)	
Y	6	OBLIST.obstacle_data	
Y	3	SVECT.horz_tracker_data	
Y(3,4)	13	RAPP_TABLE.positions	
YA	4	TRKVBL.smoothing	
YD	13	DELGEOM.hor1	
YD	13	MANGEOM.hor1(3)	
YD	3	SVECT.horz_tracker_data	
YD(3,4)	13	RAPP_TABLE.velocities	
YDE	3	SVECT.horz_tracker_data	
YDEN	4	TRKVBL.smoothing	
YDI	3	SVECT.horz_tracker_data	
YDIN	4	TRKVBL.smoothing	
YDPRJ(4)	3	SVECT.domino_obj_proj	
YL	7	CSVBL.limits	
YL	13	DOMINOVBL.coarse_screen	
YMAX	13	DOMINOVBL.coarse_screen	
YMIN	13	DOMINOVBL.coarse_screen	
YP	7	CSVBL.predictions	
YP	3	SVECT.horz_tracker_data	
YPI	3	SVECT.horz_tracker_data	
YPR(9)	13	DOMINOVBL.coarse_screen	
YPRJ(4)	3	SVECT.domino_obj_proj	
YS	4	TRKVBL.smoothing	
YSI	4	TRKVBL.smoothing	
YU	7	CSVBL.limits	
YU	13	DOMINOVBL.coarse_screen	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
Z	13	DELGEOM.ver1	
Z	13	MANGEOM.ver1(3)	
Z	13	NVGEOM.ver	
Z	3	SVECT.vert_tracker_data	
Z(5,4)	13	RAPP_TABLE.positions	
ZAFCON	8	CAPARM.zone2	275 ft
ZCARE	13	RAERPARM.multi-AC	150 ft
ZCORRECT	4	TRKPARM.vert_tracker	0.9
ZD	13	DELGEOM.ver1	
ZD	13	MANGEOM.ver1(3)	
ZD	13	NVGEOM.ver	
ZD	3	SVECT.vert_tracker_data	
ZD(5,4)	13	RAPP_TABLE.velocities	
ZDDWNF	13	MODELING.values	1500 ft/min
ZDDWNS	13	MODELING.values	800 ft/min
ZDE	3	SVECT.vert_tracker_data	
ZDFD	13	RATE.ac1	
ZDFM(3)	13	RATE.ac1	
ZDPRJ	3	SVECT.domino_obj_proj	
ZDTH	13	RAERPARM.features	6 ft/s
ZDTHR	8	BCSVBL.res	-1 ft/s
ZDTHRTA	8	BCSVBL.threat	-1 ft/s
ZDUPF	13	MODELING.values	1500 ft/min
ZDUPS	13	MODELING.values	800 ft/min
ZFAST	3	CSCREEN.thresholds	16.67 ft/s
ZHMNX	3	AZPARM.coarse_region	Table 8-6
ZHMNY	3	AZPARM.coarse_region	Table 8-6
ZHMXX	3	AZPARM.coarse_region	Table 8-6
ZHMXY	3	AZPARM.coarse_region	Table 8-6
ZJMNX	3	AZPARM.coarse_region	Table 8-7
ZJMNY	3	AZPARM.coarse_region	Table 8-7
ZJMXX	3	AZPARM.coarse_region	Table 8-7
ZJMXY	3	AZPARM.coarse_region	Table 8-7
ZL	7	CSVBL.limits	

NAME	CHAPTER	CONTEXT (STRUCTURE/GROUP)	NOMINAL VALUE
ZL	13	DOMINOVBL.coarse_screen	
ZMAX	13	DOMINOVBL.coarse_screen	
ZMCC	6	TAO.misc_variables	
ZMIN	13	DOMINOVBL.coarse_screen	
ZMX	13	DOMINOVBL.detection	
ZNOM	3	SYSTEM.tracker	5000 ft
ZP	7	CSVBL.predictions	
ZP	3	SVECT.vert_tracker_data	
ZPR(5)	13	DOMINOVBL.coarse_screen	
ZPREV	3	SVECT.vert_tracker_data	
ZPRJ(4)	3	SVECT.domino_obj_proj	
ZPRT	3	SVECT.general_values	
ZR	4	TRKVBL.vert_tracker	
ZRCON2	8	CAPARM.zone2	0.25 nmi^2
ZS	3	SVECT.vert_tracker_data	
ZSMOOTH	4	TRKPARM.vert_tracker	0.3
ZTHR	8	BCSVBL.res	1200 ft
ZTHETA	8	BCSVBL.threat	1500 ft
ZU	7	CSVBL.limits	
ZU	13	DOMINOVBL.coarse_screen	
ZVEL_INIT	4	RPTPARM.ztrk_init	1. f/s
ZZON2	3	AZPARM.arznvb	200 ft
Z7	4	TRKVBL.vert_tracker	
Z8SRC1	13	RARRVBL.res_adv	
Z8SRC2	13	RARRVBL.res_adv	

PSEUDOCODE TABLE OF CONTENTS

```
-----------------------------------------------------------------------------------------

                    <CONFLICT TABLE AND PAIR RECORD CONSTANTS>


                 <*** INTERNAL VALUES OF RESOLUTION ADVISORIES ***>


     INT $NORES;                < no resolution advisory >
     INT $NULLRES;              < null resolution advisory >

     INT $TL;                   < turn left >
     INT $TR;                   < turn right >
     INT $DTR;                  < don't turn right >
     INT $DTL;                  < don't turn left >
     INT $DTLDTR;               < don't turn left, don't turn right >

     INT SCL;                   < climb >
     INT $DES;                  < descend >
     INT $DDES;                 < don't descend >
     INT $DCL;                  < don't climb >
     INT $DCLDDES;              < don't climb, don't descend >
     INT $LDES2K;               < limit descent to 2000 ft/min >
     INT $LCL2K;                < limit climb to 2000 ft/min >
     INT $LDES1K;               < limit descent to 1000 ft/min >
     INT $LCL1K;                < limit climb to 1000 ft/min >
     INT $LDES500;              < limit descent to 500 ft/min >
     INT $LCL500;               < limit climb to 500 ft/min >


                       <*** ATSID FIELD CONSTANT ***>


     INT $BCAS;                 < BCAS in control of conflict >


                        <*** PAC FIELD CONSTANT ***>


     INT $UNK;                  < AC ID unknown >
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------

                    <*** POSCHD FIELD CONSTANTS ***>

   INT $DOUBLE;              < double dimension resolution advisories in pair
                                record >
   INT $NEG;                 < negative resolution advisories in pair record >
   INT $NORA;                < no resolution advisories are needed >
   INT $NOTSET;              < initial pair record creation >
   INT $ONEHIT;              < first requirement for resolution advisories >
   INT $ONEMIS;              < first miss for resolution advisories >
   INT $POS;                 < positive single dimension resolution advisories
                                in pair record >
   INT $RANEC;               < resolution advisories initial necessity >
   INT $RCHDBL;              < recompute double dimension resolution
                                advisories >
   INT $RCHSNG;              < recompute single dimension resolution
                                advisories >
```

```
--------------------------------------------------------------------------------
                         <DETECTION CONSTANTS>

                  <*** "SPECIAL MEANING" CONSTANTS ***>

FLT $UDMD;                  < undefined miss distance >
FLT $UDTAU;                 < undefined tau >
```

<DOMINO CONSTANTS>

<*** DOMINO PROJECTION VALUES ***>

```
FLT $UNPOS;         < uncomputed x, y, z position vectors >
FLT $UNVEL;         < uncomputed x, y, z velocity components >
```

<*** SUBJECT AC POTENTIAL RESOLUTION ADVISORY STATUS ***>

```
INT $DOMCC;         < this potential resolution advisory causes a domino
                      conflict >
INT $DOMCNC;        < this potential resolution advisory does not cause a
                      domino conflict >
INT $DOMNP;         < this is a potential resolution advisory, for which
                      domino processing has not been performed >
INT $NOTPRA;        < this is not a potential resolution advisory for this AC >
```

<*** OBJECT AC RESOLUTION ADVISORY STATUS ***>

```
INT $DOMC;          < this resolution advisory tested against this potential
                      domino conflict AC and domino conflict caused >
INT $NODOMC;        < this resolution advisory test for domino against this
                      potential domino conflict AC and no domino conflict caused >
INT $NOTTEST;       < this resolution advisory not tested for domino against
                      this potential domino conflict AC >
```

```
--------------------------------------------------------------------------------

                          <PAIR STATUS CONSTANTS>
      <(ROUTINE AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION)>

                      <*** PRCONT and PREQ VALUES ***>

  INT $BOTHCONT;          < both AC in pair controlled >
  INT $BOTHEQ:            < both AC in pair ATARS equipped >
  INT $NOCONT;            < neither AC controlled >
  INT $NOEQ:              < no AC in pair ATARS equipped >
  INT $ONECONT;           < one AC in pair controlled >
  INT $ONEEQ:             < one AC in pair ATARS equipped >
```

```
--------------------------------------------------------------------------------
                          <PSEP MODELING CONSTANTS>

                          <*** VERTICAL LEVELS ***>

     INT $LEV1   =   1;     < level 1 >
     INT $LEV2   =   2;     < level 2 >
     INT $LEV3   =   3;     < level 3 >

                          <*** HORIZONTAL PATHS ***>

     INT $TLP    =   1;     < 'turn left' path >
     INT $CSP    =   2;     < 'continue straight' path >
     INT $TRP    =   3;     < 'turn right' path >
```

```
-----------------------------------------------------------------------------

                        <STATE VECTOR CONSTANTS>

                          <*** ACAT VALUES ***>

INT $AT1;                  < AC in immediate vicinity of airfield >
INT $AT2;                  < AC along active runway final approach >
INT $AT3;                  < AC in general vicinity of sensor >
INT $AT4;                  < AC far from sensor >
INT $UNAT;                 < no area type defined >

                          <*** ACLASS VALUES ***>

INT $CL0;                  < class 0 ATARS service >
INT $CL1;                  < class 1 ATARS service >
INT $CL2;                  < class 2 ATARS service >

                          <*** ATSEQ VALUES ***>

INT $AEQ;                  < AC is ATARS-equipped >
INT $ABEQ;                 < AC is ATARS/BCAS-equipped >
INT $UNEQ;                 < AC is not ATARS-equipped >

                          <*** FAZ VALUES ***>

INT $FAZ0;                 < AC not in a final approach zone >
INT $FAZ1;                 < AC near the airfield >
INT $FAZ2;                 < AC along glide slope >
INT $UDFAZ;                < undefined final approach zone >

                          <*** TYPE VALUES ***>

INT $ATCRBS;               < ATCRBS-equipped AC >
INT $DABS;                 < DABS-equipped AC >


---------------------------- ATARS SYMBOLIC CONSTANTS ----------------------------
```

```
----------------------------------------------------------------------------------

                        <*** TURN STATE CONSTANTS ***>

   INT $STRNGLFT;                 < strong left turn indication >
   INT $WKLFT;                    < weak left turn indication >
   INT $STRAIGHT;                 < AC seems to be going straight >
   INT $WKRGT;                    < weak right turn indication >
   INT $STRNGRGT;                 < strong right turn indication >
   INT $NUMMINUS;                 < we don't know what AC is doing >
   INT $NUMPLUS;                  < we don't know what AC is doing >
```

```
---------------------------------------------------------------------------------

                      <TERRAIN/AIRSPACE/OBSTACLE CONSTANTS>

            <*** CONSTANTS USED IN BUILDING CONTROLLER ALERT MESSAGE ***>

     BITS $CAM     = 10000011;   < type code for controller alert message >
     BITS $COAT    =       11;   < indicates aircraft is controlled and
                                   ATARS-equipped >
     BITS $COUN    =       10;   < indicates aircraft is controlled and unequipped >
     BIT $NOVOICE =        0;   < indicates controller voice communication
                                   not required >
     BITS $OAM     =       10;   < message type for obstacle alert >
     BITS $RAM     =       11;   < message type for restricted airspace alert >
     BITS $TAM     =       01;   < message type for terrain alert >
     BIT $VOICE    =        1;   < indicates controller voice communication
                                   required >
```

APPENDIX C

SYNTAX OF E PSEUDOCODE

This appendix provides a concise overview of the syntax of the pseudolanguage E. The information supplied should be sufficient to allow the reader to decipher the logic specified in this document. For a complete discussion of pseudolanguage in general and E in particular, see Reference 13.

C.1 General Information

A. E = Eclectic System Specification Language

B. E is similar to other pseudolanguages, except that indentation counts: no BEGIN/END, IF/ENDIF, DO/OD, etc.

C. E character set conventions:

Underscored text denotes E constructs.
Uppercase text denotes "real" program statements.
Lowercase text denotes abstract (pseudo) statements.
Angle brackets ("<", ">") denote comments.
Semicolons are used as statement delimiters.

An example:

```
--------------------------------------------------
    REPEAT UNTIL (all conditions satisfied);
        Obtain message type;
        IF (obsolete message OR A EQ SQRT(B))
            THEN PERFORM message_elimination;
    ENDREPEAT;
--------------------------------------------------
```

D. Identifiers have no inherent length limit. Underscores are used to break up long names, as shown in the example above.

E. Syntax definitions below follow convention of having square brackets ("[", "]") indicate optional statement elements.

C.2 Blocks

External Blocks

TASKs and ROUTINEs are the external blocks supported by E. Although they are functionally equivalent, ROUTINEs tend to be subordinate to (i.e., invoked by) TASKs.

Syntax:

```
-------------------------------------------------------

     TASK taskname
          [IN (input parameter(s))]
          [OUT (output parameter(s))]
          [INOUT (modified parameter(s))];
      ...
      ...
     END taskname;

-------------------------------------------------------
```

```
-------------------------------------------------------

     ROUTINE routinename
          [IN (input parameter(s))]
          [OUT (output parameter(s))]
          [INOUT (modified parameter(s))];
      ...
      ...
     END routinename;

-------------------------------------------------------
```

Input parameters are read but not modified; output parameters are set by the block; modified parameters are read and then modified.

Functions may also be defined. The returned value may be assigned to the single output parameter or, alternatively, assigned to the function name itself.

```
     FUNCTION functionname
          [IN (input parameter(s))]
          [OUT (output parameter)];
      ...
      ...
     END functionname;
```

Invocation of External Blocks

TASKs and ROUTINEs:

```
     CALL blockname
          [IN (input parameter(s))]
          [OUT (output parameter(s))]
          [INOUT (modified parameter(s))];
```

Functions are invoked by name:

```
     J = SQRT(K);
     L = OWNER_OF(M);
```

Internal Blocks

Internal blocks (known as processes) serve as a means of decomposing a large block (external or internal) into manageable one-page segments. They are known only to the block in which they are defined. They do not accept parameters, as it is assumed that internal blocks have access to all variables known to the invoking block.

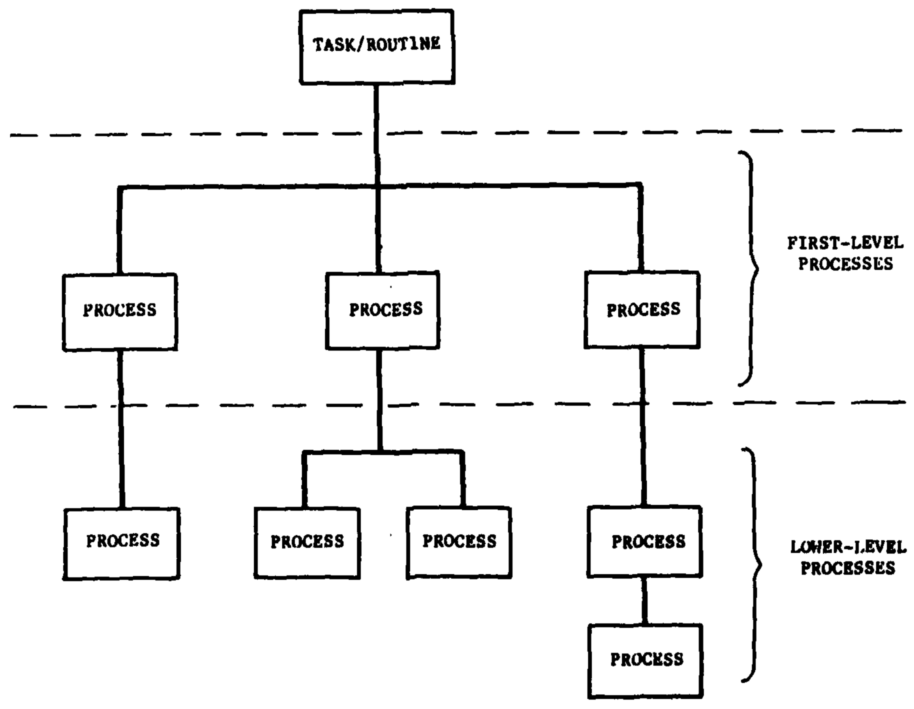
```
     PROCESS processname;
      ...
      ...
     END processname;
```

Invocation of Internal Blocks

```
    PERFORM processname;
```

Nomenclature of Internal Blocks

A TASK or ROUTINE might be decomposed into processes as follows:



Processes frequently invoke external blocks (TASKs and ROUTINEs).

Logic Document Organization

When pseudocode is presented in formal logic documents, each chapter is typically devoted to defining a single task. Within a task definition, the blocks of pseudocode appear in a standard sequence:

- The task's main logic;
- First-level processes, in order of invocation;
- Lower-level processes, in alphabetical order.

C.3 Data Types

Variables

Variables are declared at the beginning of a block in the format shown below:

```
BIT bitname;                    <single logical bit>
BITS stringname;                <bit string>
CHR stringname;                 <character string>
FLG bitname;                    <synonym for BIT>
FLT name;                       <floating point number>
INT name;                       <integer>
PTR name;                       <pointer>
```

The precision of numeric variables and the length of strings are implementation-dependent, although comments on the declaration may be used to indicate specific requirements.

Arrays are declared by means of subscripts:

```
INT ZONES(4);                   <four-element array>
```

E uses PL/I syntax for pointer qualification. Thus,

```
    P→X
```

means "the copy of X pointed to by P."

Constants

Constants in E are variables that are assigned a value when they are declared and keep that value forever. By convention, constant names are preceded by dollar signs in E to remind the reader that they are special.

```
    FLT $FTPERNM   = 6076.115;
    INT $TL        =    2;        <Turn Left>
```

On occasion, the constant is shown without a corresponding value. This convention indicates that a constant is required but that its value may be anything the system implementor chooses.

Built-in Constants

Strictly speaking, the only hard constants permitted in code are zero and one. E recognizes the logical constants $TRUE and $FALSE. Two special statements are provided to set and clear bits:

```
    SET bitname;                  <bitname = $TRUE >
    CLEAR bitname;                <bitname = $FALSE >
```

The built-in constant $NULL defines a null pointer.

Data Structures

E provides a mechanism for grouping related variables into data structures:

```
     STRUCTURE structurename
       GROUP groupname
          FLT variablename
                   ...
                   ...
     ENDSTRUCTURE;
```

An arbitrary number of groups may be defined. The keyword LIKE may be used to indicate that a group is identical to another group or a structure identical to another structure.

When a variable that has been declared inside a data structure is used in code, it must be qualified with the name of the structure (and the name of the group, if needed to resolve ambiguity):

```
     SVECT.X = PREC.ctl_thresh.ALT;
```

When groups are manipulated as a unit, the GROUP keyword may be included as an aid to the reader:

```
     CALL COMPUTE
          INOUT (GROUP SVECT.radar_reports);
```

Expressions

E assumes the existence of the usual repertoire of built-in functions (ABS, SIN, SIGN,...). Within logical expressions, logical operators are of the form LE, LT, GE, and so on.

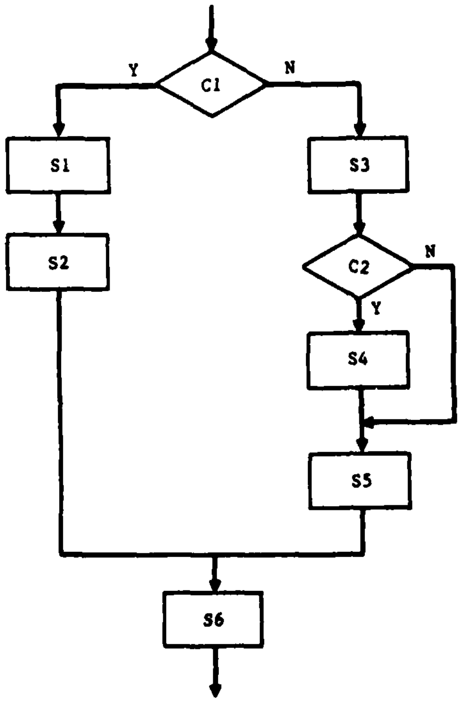
C.4 Flow-of-Control Constructs

E uses a set of flow-of-control constructs that incorporates structured programming principles. Readers familiar with other pseudolanguages are once again reminded that indentation counts.

IF-THEN-ELSE

This is the usual conditional. The ELSE clause is optional (but recommended in complex statements). Since readers will presumably be familiar with the syntax of this construct, the example below is meant to emphasize the effects of indentation.

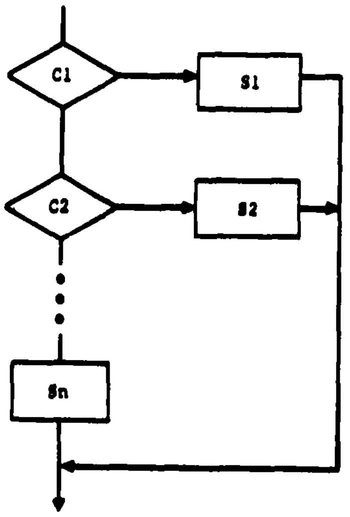
```
-------------------------

     IF (cond1)

          THEN s1;

               S2;

          ELSE s3;

               IF (cond2)

                    THEN s4;

               S5;

     S6;

-----------------------
```



IF-ELSEIF-OTHERWISE

This is the multiple-choice conditional (like SELECT-CASE in other languages). The conditions are mutually exclusive. If all the logical tests fail, the optional OTHERWISE clause is executed.

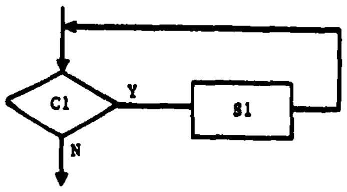
```
-----------------------------

    IF (cond1)

         THEN s1;

    ELSEIF (cond2)

         THEN s2;

          ...

          ...

    [OTHERWISE sn;]

-----------------------------
```



REPEAT-WHILE

This is the first of three looping constructs. Note that the logical test takes place at the top of the loop, so that the loop may never be executed.

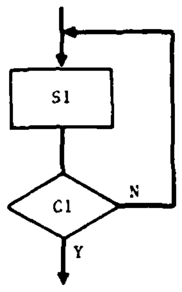
```
-----------------------------

    REPEAT WHILE (cond1);

         S1;

         ...

    ENDREPEAT;

-----------------------------
```



REPEAT-UNTIL

This construct is the complement of REPEAT-WHILE: the logical test is performed at the end of the loop and the loop continues while the condition is not true. The loop body is always executed at least once.

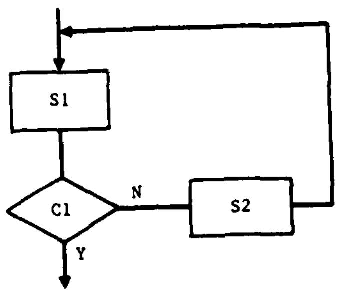
```
-----------------------------
    REPEAT UNTIL (cond1);
         S1;
         ...
    ENDREPEAT;
-----------------------------
```



LOOP-EXITIF-ENDLOOP

This construct provides a good general-purpose looping mechanism.

```
-----------------------------
    LOOP;
         S1;
    EXITIF (cond1);
         S2;
    ENDLOOP;
-----------------------------
```



In some cases, low-level operations within the three looping constructs (such as obtaining the next element in a linked list) will be omitted for brevity.

APPENDIX D

REFERENCES

1. "Engineering Requirement for a Discrete Address Beacon System (DABS) Sensor," Federal Aviation Administration, FAA-ER-240-26A.

2. J. Dieudonne, "Automatic Traffic Advisory and Resolution Service (ATARS): A Functional Description," The MITRE Corporation, McLean, Virginia, WP-80W00426, May 1980.

3. J. Dieudonne, and R. Lautenschlager, "DABS/ATARS/ATC Operational Systems Description," The MITRE Corporation, McLean, Virginia, MTR-79W00436, (Federal Aviation Administration, FAA-RD-80-42) April 1980.

4. R. H. Lentz, W. D. Love, N. S. Malthouse, D. L. Roberts, T. L. Signore, R. A. Tornese, A. D. Zeitlin, "Automatic Traffic Advisory and Resolution Service (ATARS) Multi-site Algorithms," The MITRE Corporation, McLean, Virginia, MTR-80W00100, Rev. 1, (Federal Aviation Administration, FAA-RD-80-3, Rev. 1) October 1980.

5. J. A. Grupe, R. H. Lentz, W. D. Love, A. L. McFarland, W. P. Niedringhaus, D. G. Pohoryles, N. A. Spencer, L. B. Zarrelli, and A. D. Zeitlin, "Active BCAS Detailed Collision Avoidance Algorithms," The MITRE Corporation, McLean, Virginia, MTR-80W286, October 1980.

6. N. E. Fredman, "A Study of ATARS Turn Sensing for Use in Resolution Evaluation," The MITRE Corporation, McLean, Virginia, MTR-80W00110, (Draft-May 1980).

7. "Minimum Safe Altitude Warning Design Data," Sperry Univac, St. Paul, Minnesota, PX-11325, Rev. B, March 1977.

8. J. DeMeo, "DABS/ATC Facility Surveillance and Communications Message Formats," Federal Aviation Administration, FAA-RD-80-14, ATC-33.

9. "Draft U.S. National Aviation Standard for the Automatic Traffic Advisory and Resolution Service," Federal Aviation Administration, Federal Register Volume 46, No. 58, 18885, March 26, 1981.

10. "U.S. National Aviation Standard for the Discrete Address Beacon System," Federal Aviation Administration, FAA Order 6365.1, December 1980.

11. "U.S. National Aviation Standard for the Active Beacon Collision Avoidance System," Federal Aviation Administration, October 1980.

12. J. Andrews, "An Improved Technique for Altitude Tracking of Aircraft," FAA-RD-80-139, Lincoln Laboratory, ATC-105, (Draft - 20 January 1981).

13. H. R. Bulterman, "All About E," The MITRE Corporation, McLean, Virginia, WP-80W00654, (Draft-August 1980).

*U.S. GOVERNMENT PRINTING OFFICE: 1981-341-428/1276--1-3